Rule Learning in Knowledge Graphs

Author:
Pouya Ghiasnezhad Omran

Supervisors:
Professor Kewen Wang
Dr Zhe Wang

Pouya Ghiasnezhad Omran
BEng, MSc

Institute for Integrated and Intelligent Systems
School of Information and Communication Technology
Griffith University

Submitted in fulfillment of the requirements of the degree of Doctor of Philosophy

October 2018
To my Family.
Abstract

With recent advancements in knowledge extraction and knowledge management systems, an enormous number of knowledge bases have been constructed, such as YAGO, and Wikidata. These automatically built knowledge bases which contain millions of entities and their relations have been stored in graph-based schemas, and thus are usually referred to as knowledge graphs (KGs).

Since KGs have been built based on the limited available data, they are far from complete. However, learning frequent patterns in the form of logical rules from these incomplete KGs has two main advantages. First, by applying the learned rules, we can infer new facts, so we could complete the KGs. Second, the rules are stand-alone knowledge which express valuable insight about the data.

However, learning rules from KGs in relation to the real-world scenarios imposes several challenges. First, due to the vast size of real-world KGs, developing a rule learning method is challenging. In fact, existing methods are not scalable for learning first order rules, while various optimisation strategies are used such as sampling and language bias (i.e., restrictions on the form of rules). Second, applying the learned rules to the vast KG and inferring new facts is another difficult issue. Learned rules usually contain a lot of noises and adding new facts can cause inconsistency of KGs. Third, it is useful but non-trivial to extend an existing method of rule learning to the case of stream KGs. Forth, in many data repositories, the facts are augmented with time stamps. In this case, we face a stream of data (KGs). Considering time as a new dimension of data imposes some challenges to the rule learning process. It would be useful to construct a time-sensitive model from the stream of data and apply the obtained model to stream KGs. Last, the density of information in a KG is varied. Although the size of a KG is vast, it contains a limited amount of information for some relations. Consequently, that part of KG is sparse. Learning a set of accurate and informative rules regarding the sparse part of a KG is challenging due to the lack of sufficient training data.

In this thesis, we investigate these research problems and present our methods for rule learning in various scenarios. We have first developed a new approach, named
Rule Learning via Learning Representation (RLvLR), to learning rules from KGs by using the technique of embedding in representation learning together with a new sampling method. RLvLR learns first-order rules from vast KGs by exploring the embedding space. It can handle some large KGs that cannot be handled by existing rule learners efficiently, due to a novel sampling method. To improve the performance of RLvLR for handling sparse data, we propose a transfer learning method, Transfer Rule Learner (TRL), for rule learning. Based on a similarity characterised by the embedding representation, our method is able to select most relevant KGs and rules to transfer from a pool of KGs whose rules have been obtained. We have also adapted RLvLR to handle stream KGs instead of static KGs. Then a system called StreamLearner is developed for learning rules from stream KGs. These proposed methods can only learn so-called closed path rules, which is a proper subset of Horn rules. Thus, we have also developed a transfer rule learner (T-LPAD) that learns the structure of logic program with annotated disjunctions. T-LPAD is created by employing transfer learning to explore the space of rules’ structures more efficiently.

Various experiments have been conducted to test and validate the proposed methods. Our experimental results show that our methods outperform state-of-the-art methods in many ways.
Statement of Originality

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Pouya Ghiasnezhad Omran
List of Publications

The main contributions of this study are either published or got accepted to the reputable conferences in the field.


- Pouya Ghiasnezhad Omran, Kewen Wang, Zhe Wang: Learning Temporal Rules from Knowledge Graph Streams. In the AAAI Spring Symposium Series 2019 [accepted for publication]

- Pouya Ghiasnezhad Omran, Zhe Wang, Kewen Wang: Knowledge Graph Rule Mining via Transfer Learning. In PAKDD 2019 [accepted for publication]
Acknowledgement

This work is the result of my journey throughout a sea of difficulties and enjoyments. I would like to express my gratefulness to everyone who accompanied me on this course.

I would like to express my deep appreciation to my supervisor, Professor Kewen Wang, for his kind support and guidance during the course of Ph.D. program; he arranged a friendly and professional research atmosphere and instructed me with his brilliant scientific vision. I also want to thank my co-supervisor Dr. Zhe Wang who continuously guided and supported me. I greatly appreciate his advices and suggestions which kept me in the proper research path.

Lastly and most importantly, I would like to thank my wife Mahtab who lightened the night even in the thickest darkness. Also, thanks to my family members for their unconditional love and support especially to my dear brother Nima.
Contents

1 Introduction
  1.1 Contributions .................................................. 6
  1.2 Structure of the Dissertation ................................ 8

2 Background
  2.1 Modeling Knowledge Graphs .................................... 10
     2.1.1 Rule-based methods for Modeling KGs .................... 13
     2.1.2 Embedding-Based Methods for Modelling KGs .......... 16
     2.1.3 Hybrid Methods ............................................ 18
  2.2 Learning Logical Models from KG Streams .................... 20
     2.2.1 KG Streams and Temporal Rules ......................... 20
     2.2.2 KG Streams Reasoning and Learning ................. 21
  2.3 Transfer Learning for Knowledge Bases ..................... 22
  2.4 Logic and its Probabilistic Extensions ...................... 24
     2.4.1 Probabilistic Logical Model ............................. 25

3 Learning Rules via Learning Representation from KGs .......... 29
  3.1 Introduction .................................................... 30
  3.2 An Overview of Our Approach ................................ 32
  3.3 Scoring Functions .............................................. 34
  3.4 Rule Evaluation ................................................ 36
  3.5 Experiments .................................................... 38
     3.5.1 Rule Mining ............................................... 40
     3.5.2 Link Prediction ............................................ 42
  3.6 Conclusion ..................................................... 44

4 Learning Temporal Rules from KG Streams ....................... 46
  4.1 Introduction .................................................... 47
  4.2 Temporal Rule Learning ....................................... 48
  4.3 Combine Stream Learning and Reasoning .................... 51
  4.4 Experiments .................................................... 54
  4.5 Related Works ................................................ 61
  4.6 Conclusion ..................................................... 61

5 Learning Rules via Transfer Learning from KGs .................. 65
5.1 Introduction ............................................. 66
5.2 An Overview of Our Approach ........................ 67
5.3 Similarity Measure via Embeddings ................. 71
   5.3.1 Similarity Between Goal Predicates ............. 71
   5.3.2 Similarity Between Other Predicates ............ 73
5.4 Experiments ........................................... 75
   5.4.1 From FB15K to YAGO2s .......................... 76
   5.4.2 From FB15K to FB15K ............................ 78
5.5 Related Works .......................................... 80
5.6 Conclusion ............................................. 81

6 Learning Probabilistic Rules via Transfer Learning from KBs 82
   6.1 Introduction ........................................... 83
   6.2 An Overview of Our Approach ......................... 86
      6.2.1 PD Graph and Similarity Matrix .................. 87
      6.2.2 Rule Construction ................................ 90
   6.3 Experiments ........................................... 92
   6.4 Related Works ......................................... 95
   6.5 Conclusion ............................................. 96

7 Conclusion and Future Work ............................ 97
   7.1 Results Summary ..................................... 98
   7.2 Discussion and Future Work .......................... 100

Bibliography ............................................... 104
List of Figures

3.1 The paths corresponding for a rule. .................................. 37
4.1 Example of KG stream.................................................. 47
4.2 Stream learning and reasoning...................................... 53
4.3 Performance comparison on link prediction over KG streams... 63
4.4 Performance over varying window sizes............................ 64
4.5 Performance over varying minimum prediction distances........ 64
5.1 Mappings based on translation and linear transformation........ 75
6.1 PD graphs for two domains UWC-SE and IMDB..................... 88
# List of Tables

3.1 Benchmark specifications ......................................................... 39
3.2 Rule mining comparison between RLvLR and AMIE+ ......................... 40
3.3 The numbers of new facts predicted by RLvLR and AMIE+ ................. 41
3.4 Number of mined rules with different scoring functions .................... 42
3.5 Link prediction of RLvLR and Neural LP on large KGs ....................... 44
3.6 The comparison on link prediction on FB15K-237 .......................... 44
4.1 Dataset statistics ................................................................. 56
4.2 Temporal rules in link prediction .............................................. 57
4.3 Performance on varying sizes of structure learning data .................... 59
5.1 Benchmark specifications ........................................................ 75
5.2 Transfer rule learning from large KGs to small KGs ........................ 77
5.3 Transfer rule learning from dense predicates to sparse predicates ........ 79
5.4 Comparison of different similarity measures .................................. 80
6.1 Comparison between T-LPAD and SLIPCOVER .............................. 94
6.2 Training times with 6 folds training data ..................................... 94
Chapter 1

Introduction

Many efforts have been made recently in the construction of large knowledge bases (KBs) that include millions of facts about various entities in the world such as, people, universities, movies, animals, etc. These knowledge bases have been shown to be incredibly beneficial for intelligent Web searching, question understanding, in-context advertising, social media mining, and biomedicine. Due to their new features, such modern knowledge bases are often referred to as knowledge graphs or just KGs. Major examples of KGs include YAGO [1], DBpedia [2], Wikidata [3] and Freebase [4].

Due to their massive data volume, it is impossible to construct KGs manually. Thus, automatic or semi-automatic methods are needed to construct such KGs. Despite the extensive work in automatic knowledge graph creation, current knowledge graphs are far from complete. Besides, although these KGs contain million of facts which present detailed knowledge about specific objects, they do not contain any general knowledge which gives insight into the patterns which are held in the whole KG. To address both of these issues it is critical to develop methods to acquire abstract knowledge (e.g. logical models) from such KGs.

A promising research direction is using the methods from Knowledge Representation (KR) to formalize certain predictive models and to infer new facts by applying such models, along with deploying the approaches from Machine Learning (ML)
to effectively learn the predictive model from the raw data. Therefore, the aim of our research is to develop several methods to effectively address the problem of learning predictive models in the form of logical rules from KGs. Our techniques generalize and advance relevant methods in KR and ML.

In this thesis, the main focus is on the learning of logical human-understandable models from KBs under different application scenarios. Despite the popularity of statistical approaches in the data modeling tasks such as deep learning and representation learning, we focus on the logic-based modeling due to the transparency and human-understandability of these models. Besides, pursuing logic-based learning approaches enables us to deploy techniques and tools in Knowledge Representation such as inference methods along with ML techniques like representation learning and transfer learning.

Learning logical rules from KGs involves different research problems. We consider the following four critical research problems due to their prevalence in the many real-world applications. First, the vast size of KG makes the learning and inferencing tasks more challenging because the learning/inferencing system should scale up regarding the size of KG. Second, in many real-world applications, the time of a fact being valid/true is an essential feature of data which cannot be ignored. In this case, the data should be processed as it is perceived, so learning and reasoning need to be done continuously through the stream of incoming data. Consequently, a time-sensitive model like temporal rules should be learned from the stream of KGs. Third, although the size of KG is typically vast, the KG might contain sparse data regarding a number of relations. In this case, the lack of sufficient training data is a critical issue for the learning procedure. Fourth, to capture the data more accurately, probabilistic logical rules are proposed. Learning such hybrid models is a challenging task since the learner should determine both the structure and probabilistic features of rules. Consequently, we study these four different learning problems as elaborated in the following. For each of the studied problems, a specific learning system is developed to handle the corresponding learning task.
In general, this thesis aims to address the problem of logical models learning from knowledge bases. First, we presume vast KGs as the input, and the developed system learns first-order rules from scratch by using representation learning techniques. Second, we consider learning temporal rules to address the stream learning task in KG streams. Third, we address the sparsity of data in KGs with transfer learning and representation learning together. Finally, we employ knowledge bases including facts with arbitrary arity as the input (In contrast with the KG which contains facts with binary arity) and build a transfer learner that uses this type information to learn probabilistic logical rules.

**Learning rules from vast KGs** In Knowledge Graphs the amount of data is enormous. Learning patterns from such KGs to capture frequent logical patterns in the data as logical rules is a critical task. Consider the following rule, \( \text{livesIn}(x, y) \land \text{locatedIn}(y, z) \rightarrow \text{isCitizenOf}(x, z) \).

This rule entails that if a person lives in a city and that city is located in a country, the person is likely to be a citizen of the country. Such a rule has a confidence degree which indicates the partition of cases in which this rule is valid in the KG. Thus, such a rule can capture the frequent patterns precisely and reflect the uncertainty of patterns in the real-world data.

The learned rules can be deployed in the following applications: (i) Since the KGs are far from complete in the Knowledge Graph Completion (KGC) task, a learner should predict the missing facts by considering the given data. The KGC task can be addressed by applying the rules to the given data and inferring the new facts. (ii) the rules are stand-alone knowledge which explains the general patterns in the KG and improves our understanding of KG.

Developing an algorithm that can learn rules from such a vast amount of data is a challenging task. There are some rule miners in the literature which address this problem including SWARM [5], RDF2rules [6], ScaleKB [7] and AMIE+ [8]. However, these systems need to be improved regarding their efficiency and accuracy. For example, for WikiData AMIE+ can only learn rules with length up
to three but cannot learn more extended rules in a reasonable time [8]. In our first research task, we address this problem by proposing a method, named Rule Learning via Learning Representation (RLvLR), which deploys the representation learning concept and a novel sampling method. RLvLR uses the embedding representation of KG elements (i.e., predicates and entities) as a heuristic to guide the search of the rule learning procedure.

**Learning temporal rules from KG streams** In many real-world data, the facts are augmented with the times of facts being valid/true. In other words, time is a new dimension of each fact. In many cases such as dealing with data with an evolving nature (i.e., the facts are collected incrementally through the time), it is useful to view it as a stream of KGs, because the learning and reasoning are needed from the early stages and the learner requires to continuously adapt its model according to new perceived data. Besides, with the stream view of data, we could keep a restricted range of data regarding previous time. This feature is helpful when the massive amount of historical data overwhelms the learner. In this realm, the learning system perceives the data as a stream, so all the data is not available to the system. For example, ICEWS [9] is a framework for early warning designed to assist US policy experts. In this framework, the temporal KG is available which can be used to predict a range of international crises. By considering this temporal KG as a stream of KGs, the system could learn and reason continuously through the upcoming data instead of considering all the available data.

To manage the KG streams, a few formalisms have been proposed for expressing the temporal rules which are held in such temporal KGs [10, 11]. Although these methods address the reasoning task in the time-augmented data, the task of learning logical models from the KG streams remains open and unaddressed.

In this research problem, we address the learning and reasoning in KG streams by developing a method, named StreamLearner. StreamLearner learns temporal rules from KG streams and infers new facts by applying the learned rules. To
achieve this, we extend the existing rule quality measures to handle rule learning from time-sensitive data.

**Learning rules from sparse KGs** Despite the large sizes of KGs, the issue of data sparsity exists; that is, for many predicates a small number of relevant facts are available. For instance, only a few facts are available for the predicates, `organismsOfThisType` and `exhibitionSponsor` in Freebase (FB15k). The lack of sufficient facts to learn a predictive model regarding these predicates is a critical challenge.

Performing an effective learning procedure for dealing with the sparse KGs is the objective of our third research problem. Transfer learning methods have been proved to handle the sparsity of data effectively [12]. In transfer learning an underlying assumption of learning should be relaxed; the train and test data are taken from the same domain. In the transfer learning scenario, the learner makes use of data from other domains, as well as from the main domain. However, adopting data from different domains (source domains) to help the learning process in the main domain (target domain) is a challenging task.

Therefore, in our developed system, Transfer Rule Learner (TRL), we propose a novel transfer learning technique which can handle the learning task even if the sparsity only exists in part of a large-scale KG. To transfer the knowledge, we use the embedding representation of elements of the source and target domains.

**Learning probabilistic rules** Logic Programs with Annotated Disjunctions (LPADs) is a formalism which augments logic with probabilistic features. Considering the probabilistic features for learned rules enables us to capture the patterns of data more precisely. However, this advantage comes with a cost; learning complexity. In this case, the cost of the probabilistic parameters learning adds to the cost of learning the structures of rules.
Several approaches regarding the learning probabilistic logic rules have been proposed in the literature [13]. However, most learning systems based on these approaches are not efficient for handling practical problems (especially, in the case of structure learning). It has been a challenging issue to reduce the search space of candidate probabilistic logic rules.

Regarding the input data of the learning system in this research problem, we consider a more general kind of data in comparison with KG, Knowledge Bases (KBs) which contain facts with arbitrary arity while KGs merely contain facts with binary arity.

In the fourth research problem, we develop a novel transfer learner (T-LPAD) which deploys type information to transfer knowledge from the source domain to the target domain.

1.1 Contributions

Contributions of this thesis are categorized in the following four sections. These contributions aim to effectively handle the learning of the logical rules from the KBs to address the research challenges discussed above.

Rules via Learning Representation from KGs We study the problem of learning first-order rules from large Knowledge Graphs (KGs). As our first contribution, we present a new approach, RLVLR, that uses the technique of embedding in representation learning together with a new sampling method in order to learn rules from KGs. Experimental results show that our system outperforms some state-of-the-art systems. Specifically, for massive KGs with hundreds of predicates and over 10M facts, RLVLR is much faster and can learn many more quality rules than major systems for rule learning in KGs such as AMIE+ [8]. We also use the RLVLR-mined rules in an inference module to carry out the link prediction task. In this task, RLVLR outperforms Neural LP [14], a state-of-the-art link prediction system, in both runtime and accuracy.
Learning Temporal Rules from KG Streams Since existing KG rule learners are not suitable for KGs with constantly evolving data, our second contribution is the first attempt to address the problem by presenting an approach to learning temporal rules from KG streams. The learned temporal rules can be then applied in the link prediction and the event prediction over KG streams. Based on the proposed method, a system StreamLearner has been implemented. Our experimental results show that StreamLearner is effective and efficient in learning temporal rules on real-life datasets and significantly outperforms some state-of-the-art systems that do not account for temporal knowledge or evolving data.

Learning Rules via Transfer Learning from KGs Since learning rules from sparse knowledge graphs is an essential yet challenging task, our third contribution is to develop a transfer learning approach for KG rule learning that is able to select most relevant KGs and rules to transfer through the similarity principle characterised by the embedding representations. We conduct experimental evaluation on standard KGs, and the results show that our system can obtain quality rules even with extremely sparse data and its predictive accuracy outperforms the state-of-the-art rule learners (AMIE+ [8] and RLvLR) and link prediction systems (HOLE [15] and TransE [16]).

Learning Probabilistic Rules via Transfer Learning from KBs The existing approaches regarding the learning probabilistic logic programs are limited in efficiency for handling practical problems (especially, in the case of structure learning). Thus, our fourth contribution is to develop a new algorithm T-LPAD for structure learning of LPADs by employing transfer learning. The new algorithm has been implemented, and our experimental results show that T-LPAD outperforms SLIPCOVER [13] for most benchmarks used in related systems.
1.2 Structure of the Dissertation

In Chapter 2, we introduce some background knowledge for understanding the thesis. We introduce the formal definition of KG and the KG modeling task. This chapter also covers different approaches to accomplishing this task. Besides, we define the Knowledge Graph stream and review some approaches which have been proposed to analyze this kind of stream of data. Furthermore, we explain the usage of transfer learning in learning from knowledge bases and review some works in this domain. Finally, we review the probabilistic extensions of the first-order logic specifically LPAD and the existing learning system for it.

In Chapter 3, we propose a novel system for learning first-order rules from KGs by using the Representation Learning methods. We evaluate our system on vast real-world KGs such as YAGO2. We assess our system as being able to handle two tasks, rule learning and link prediction. This chapter also resulted in the following publication:


In Chapter 4, we propose a novel logic-based approach to handle the stream of data which is stored in the KG streams. We propose novel rule quality measures which are sensitive to the time. We assessed our system against benchmark ICEWS. The content of this chapter has also been presented as follows:

Pouya Ghiasnezhad Omran, Kewen Wang, Zhe Wang: Learning Temporal Rules from Knowledge Graph Streams. In the AAAI Spring Symposium Series 2019 [accepted for publication]

In Chapter 5, we introduce a novel approach to handle the sparsity of KGs by the transfer learning technique. Our method utilizes the embedding representation of domains’ elements to transfer the knowledge from a previously learned task to a new task. We assessed the system regarding different sparsity scenarios against
real-world benchmarks such as Freebase and YAGO. This following paper is based on this chapter:

Pouya Ghiasnezhad Omran, Zhe Wang, Kewen Wang: Knowledge Graph Rule Mining via Transfer Learning. In PAKDD 2019 [accepted for publication]

In Chapter 6, we propose a novel approach for learning Logic Programs with Annotated Disjunctions (LPAD) from KBs by transfer learning. This chapter resulted in the following publication:


Chapter 7 summarizes the proposed contributions in this thesis and provides a perspective on the future research.
Chapter 2

Background

In this chapter, the background of our research is presented. The Chapter is divided into four sections. In the first section, the scope of the knowledge graph modeling problem and its specifications are discussed. Also, some of the existing approaches for knowledge graph modeling are reviewed. In the second section, the notion of knowledge graph stream and some works on reasoning and learning in KG streams are presented. In the third section, we explain how transfer learning methods can be used in the learning procedure of logical models from Knowledge Bases; we also review some proposed state-of-the-art methods. In the last section, the basics of probabilistic logical models are discussed.

2.1 Modeling Knowledge Graphs

A knowledge graph is an extensive set of curated facts such as YAGO [1], DBpedia [2], and Wikidata [3]. A KG includes binary facts in the form of subject-predicate-object. This set of facts (KG) presents a directed labeled graph in which a fact is a labeled edge (predicate is the label) from a subject entity to an object entity. KGs are a special kind of Knowledge Base (KB), and a KB [17] is a general
form of knowledge repository which does not impose any specific restrictions regarding the facts schema and abstraction level of knowledge. In fact, a KB might contain general rules as well as facts.

Formally a KG contains facts about entities. An entity \( e \) is a thing such as a place, a person, etc.; a fact is an RDF triple \((e, P, e')\). That is, the entity \( e \) is related to another entity \( e' \) via the binary predicate \( P \). Following the convention in knowledge representation, we denote such a fact as \( P(e, e') \). A knowledge graph \((KG)\) is a pair \( K = (E, F) \), where \( E \) is the a set of entities and \( F \) is a set of facts.

Since large KGs are far from being complete, inference of the missing facts from a given KG in order to complete it is an essential but challenging task. To do so, modeling KGs as predictive models has been proposed. In this process, a predictive model is learned from the KG and by deploying the learned model the new facts are inferred. For example, a rule learner such as AMIE+ [8] learns a set of first-order rules as a predictive model, then applies the learned rules to the KG to infer the new facts and complete the KG.

The existing methods of KGs modeling can be categorized into the three major groups: rule-based methods, embedding-based methods, and Path Rank Algorithm (PRA) methods. These groups differ based on the type of predictive models which they learn. The learned predictive models of these methods are a set of first-order rules, a set of low-ranked vectors (embeddings), and a logistic regression model, respectively [18].

Regarding the rule-based class of methods, the Inductive Logic Programming (ILP) can be deployed to learn first-order rules from relational data [19–22]. ILP is a mature field; however, its application to mine the logical rules from KGs encounters two challenges, the lack of scalability and the need for counterexamples; these two issues will be elaborated in details later in section 2.1.1. In the other line of research, in recent years some rule mining methods have been developed to perform the task of modeling a KG as logical rules including AMIE+ [8] and ScaleKB[7]. These two methods address the challenges of learning rules from KG by optimized query writing and partitioning method.
A promising class of methods for KG modeling is embedding-based methods. In this class of methods, the KG is embedded into low-dimensional vector spaces [15, 16, 18, 23–27]. To be more specific, given a KG, entities and predicates are represented in a low-dimensional vector space (embeddings), and a scoring function is defined based on the learned embeddings to measure the plausibility of each triple. Learning and operating on latent representations enable these methods to capture some unobservable but intrinsic properties of entities and their relations.

Although these methods can learn robustly from data, in contrast with rule-based methods, they cannot use common-sense knowledge, such as the rules which play a crucial role in the maintenance of the KGs [28]. Besides, the lack of scalability is another issue as existing embedding-based approaches struggle to handle real-world KGs such as YAGO2s [8]. Furthermore, these methods do not provide any transparency; that is, they make predictions, but they do not generate general knowledge to explain the causes leading to the predictions.

The Path Ranking Algorithm (PRA) [29, 30] is another class of methods for modeling KGs, by computing feature matrices over neighbor nodes in the graph. This class of methods is similar to the rule-based methods since it uses feature spaces that contains a restricted class of Horn rules found in the graph[18]. However, this class of methods is different from rule-based methods in terms of its learning and predicting procedures (i.e. it uses Horn clause rules as features of a logistic regression model.). Scalability is a critical concern for this class of methods since it cannot handle the vast benchmarks used by rule-based methods [29, 30] for evaluation.

The main application of these models is to carry out the Knowledge Graph Completion (KGC) task. Knowledge graph completion is concerned with predicting the existence (or probability of existence) of edges (facts) in the graph based on the given edges (facts). This task is essential as the knowledge graphs are often incomplete (i.e., missing many facts), and inconsistent (i.e., some of the already existing edges are incorrect).
In the rest of this section, the focus is on the two class of methods most relevant to our research; rule-based and embedding-based classes and different proposed methods regarding these two groups are reviewed. Existing embedding-based methods and rule-based methods can complement each other. The embedding-based methods have superior accuracy with regards to the KGC task but the lack of scalability and transparency of the learned models are two concerns of this class of methods. On the other hand, existing rule-based methods address these two issues while their performance on KGC is not satisfactory in term of accuracy.

2.1.1 Rule-based methods for Modeling KGs

A Horn rule or just a rule $r$ is of the form

$$a_1 \land \ldots \land a_n \rightarrow a$$

where $a, a_1, \ldots, a_n$ are atoms of the form $P(t_1, t_2)$ with $P$ being a binary predicate and each of the $t_1$ and $t_2$ being an entity or a variable. For example, $\text{livesIn}(\text{John}, x)$ or $\text{livesIn}(y, x)$. Intuitively, the rule $r$ reads that if $a_1, \ldots, a_n$ hold, then $a$ holds too. Atom $a$ is referred to as the head of $r$, denoted $\text{head}(r)$, and atoms $a_1, \ldots, a_n$ as the body of $r$, denoted $\text{body}(r)$.

The following rule is an example of Horn rules:

$$\text{livesIn}(x, t) \land \text{bornIn}(x, z) \rightarrow \text{isCitizenOf}(x, z)$$

In many rule learning systems a language bias is used to restrict the Horn rules [7, 8, 31]. Accordingly, we use closed path as language bias in this research.

Closed path (CP) rule is a fragment of Horn rules. CP rules provide a balance between the expressive power of the mined rules and the efficiency of the rule mining task. Such a syntactic restriction is a standard approach in the rule mining literature. CP rules are the underlying formalism of Path Ranking Algorithms [30], RuleEmbedding [26], [6] and ScaleKB [7].
A CP rule (or simply a rule) $r$ is of the form

$$P_1(x, z_1) \land P_2(z_1, z_2) \land ... \land P_n(z_{n-1}, y) \rightarrow P_t(x, y).$$  

(2.3)

where $x$, $y$ and $z_i$'s are variables, each $P(u, v)$ is called an atom, and $u$ and $v$ are named the subject and object argument for $P$ respectively. Intuitively, the rule $r$ reads that if $P_1(x, z_1), P_2(z_1, z_2), ..., P_n(z_{n-1}, y)$ hold, then $P_t(x, y)$ holds too. The atom $P_t(x, y)$ is the head of $r$, denoted $\text{head}(r)$, and the set of atoms $P_1(x, z_1), P_2(z_1, z_2), ..., P_n(z_{n-1}, y)$ is the body of $r$, denoted $\text{body}(r)$. The rule $r$ is called CP as the sequence of predicates in the rule body forms a path from the subject argument to the object argument of the head predicate. Note that, CP rules allow recursion (i.e. the head predicate can occur in the body).

The rule in (2.2) is not a CP rule since the $\text{livesIn}(x, t)$ is not connected to other atoms via variable $t$. An example of the CP rule is presented in the following.

$$\text{livesIn}(x, t) \land \text{locatedIn}(t, z) \rightarrow \text{isCitizenOf}(x, z)$$  

(2.4)

Mining rules from a dataset is the central task of inductive logic programming (ILP). ILP methods induce logical rules from ground facts [20, 32, 33]. For example, ALEPH is an ILP system aims to learn rules from relational data via inverse entailment [19]. However, classical ILP systems cannot be adopted for use in large scale KGs for two reasons: First, they usually require negative facts; whereas KGs merely include positive facts. There are different paradigms for the interpretation of non-existing facts, including closed world assumption (CWA) and open world assumption (OWA). In the CWA, all the missing facts are assumed as negative facts while in the OWA the missing facts are unknown (neither positive nor negative). Due to the Open World Assumption (OWA) of KGs, absent statements cannot serve as negative facts straightforwardly. Second, the ILP methods are not capable of handling the huge amount of data in KGs. In [8], it is shown that ALEPH has difficulty in handling (relatively small) KGs such as YAGO2, since it takes more than a day to learn rules regarding some predicates.
To address the aforementioned issues, a number of KG rule miners such as SWARM [5], RDF2rules [6], ScaleKB [7], AMIE+ [8] and [31, 34–36] have been proposed.

In rule-based methods, a critical issue is the definitions of the quality measures of rules. These measures determine the objective of a learner since the learner tries to learn the rules with the highest quality. To assess the quality of the mined rules, we review measures that are used in some major systems [7, 8].

Formally, assume the head of $r$ is of the form $P(x, y)$, a pair of entities $(e, e')$ satisfies the body of $r$ in KG $\mathcal{K}$, denoted $\text{body}(r, e, e', \mathcal{K})$, if there is a way of substituting variables in the body of $r$ with entities in $\mathcal{K}$ such that (i) $(e, e')$ substitutes $(x, y)$ and (ii) all atoms in the body of $r$ (after substitution) are facts in $\mathcal{K}$. $(e, e')$ satisfying the head of $r$ in KG $\mathcal{K}$ is defined in the same way and denoted $\text{head}(r, e, e', \mathcal{K})$. Then the support degree of $r$ is defined as

$$\text{supp}(r, \mathcal{K}) = \#(e, e') : \text{body}(r, e, e', \mathcal{K}) \land \text{head}(r, e, e', \mathcal{K})$$ (2.5)

To normalize this degree, the degrees of standard confidence (SC) and head coverage (HC) are defined as follows:

$$\text{SC}(r, \mathcal{K}) = \frac{\text{supp}(r, \mathcal{K})}{\#(e, e') : \text{body}(r, e, e', \mathcal{K})}$$ (2.6)

$$\text{HC}(r, \mathcal{K}) = \frac{\text{supp}(r, \mathcal{K})}{\#(e, e') : \text{head}(r, e, e', \mathcal{K})}$$ (2.7)

In the following of this subsection, we review two state-of-the-art rule learners, AMIE+ [8] and ScaleKB [7]. These two methods are the state-of-the-art rule learners regarding the number of learned rules and the maximum size of KGs which they can handle.

AMIE+ [8] addresses two challenges of learning logical rules from KGs, lack of counterexamples and the massive size of KG. AMIE+ goal is mining closed-rules from KGs with more than 12M statements (facts). It handles the KGs such as YAGO, Wikidata, and DBpedia.
AMIE+ uses closed-rules language bias regarding the mined rules. The closedness means all variables in a rule appear at least twice. For example, $P_1(x, y) \land P_2(y, z) \rightarrow P_3(y, z)$ is not a closed rule because $x$ has only one occurrence. The restriction of closed-rules disallows the learner from mining rules with free variables. Although AMIE+ uses a closed-rule formalism which is slightly more expressive than the CP formalism, it mines rules with at most two body atoms from real-world KGs, such as YAGO2s, in a reasonable time. Thus, it excludes the more extended rules (rules with more body atoms) which might contain essential knowledge.

The key novel component of AMIE+ is using the Partial Completeness Assumption (PCA). By this assumption, AMIE+ intends to estimate the counterexamples more accurately in comparison with other systems that use CWA.

ScaleKB [7] mines CP rules from KGs. The main advantage is the handling of large-scale KGs such as Freebase with 112M entities and 388M facts. It uses several optimization methods including an early detection mechanism for low-quality rules, a parallel mining algorithm, and a partitioning algorithm to divide the vast set of facts.

ScaleKB uses CWA since it treats all missing facts as negative facts in their main experiments. This method works well according to the accuracy of their predictions. However, it mines rules with at most two body atoms from the real-world KGs such Wikidata in reasonable time, so it excludes many interesting more extended rules. ScaleKB also implements a scalable inference algorithm to apply the ScaleKB-learned rules to the KGs and infer new facts.

2.1.2 Embedding-Based Methods for Modelling KGs

Another class of KG modeling methods is embedding-based methods. These methods deploy representation learning techniques to learn a low-dimensional vector (an embedding) for each element of KGs (an entity or a predicate).

Various methods have been proposed to construct embeddings (e.g. [27, 37]), two main categories of methods are translation based embeddings [16, 24, 38–43] and
compositional embeddings \cite{15, 23}. The translation based embeddings use additive calculus and vectors to represent the embeddings of predicates. The compositional embeddings use product calculus and the embedding of a predicate is a matrix.

A method for representation learning from KGs often consists of two main steps: (1) to embed the entities and predicates of the given KG into a latent space, and (2) to reconstruct the KG based on the obtained embeddings to predict new facts.

In the rest of this subsection, we review a representative system from each of these two categories.

RESCAL \cite{23} embeds each entity $e_i$ to a vector $E_i \in \mathbb{R}^d$ and each predicate $P_k$ to a matrix $P_k \in \mathbb{R}^{d \times d}$ where $\mathbb{R}$ is the set of real numbers and $d$ is an integer (a parameter of the learner). For each given fact $P_0(e_1, e_2)$, the following scoring function is computed:

$$f(e_1, P_0, e_2) = E_1^T \cdot P_0 \cdot E_2$$  \hspace{1cm} (2.8)

The score function indicates the plausibility of the fact that $e_0$ has relation $P_0$ with $e_1$. The two sets of embeddings, \{\text{\textbf{E}}_i\} and \{\text{\textbf{P}}_k\} are learned by minimizing a reconstruction loss as follows:

$$\min_{\{\text{\textbf{E}}_i\}, \{\text{\textbf{P}}_k\}} \sum_k \sum_i \sum_j \|y_{ij}^k - f(e_i, P_k, e_j)\|^2_F + \lambda R$$ \hspace{1cm} (2.9)

where $R$ is the following regularization term

$$R = \sum_i \|e_i\|^2_F + \sum_k \|P_k\|^2_F$$ \hspace{1cm} (2.10)

and $y_{ij}^k$ is one if $(e_i, P_k, e_j) \in F$ and zero otherwise.

In conclusion, RESCAL reconstructs the KG by using the product as the composition calculus. RESCAL captures rich interactions since it learns a larger number of parameters in comparison with the methods which embed the predicates into vectors \cite{15}.
TransE [16] is the most representative translation based method. It embeds both entities \((e_i)\) and predicates \((P_k)\) to vectors in the same space, \(E_i, P_k \in \mathbb{R}^d\). For a fact \((e_1, P_0, e_2)\), the predicate \(P_0\) translates the \(e_1\) to the \(e_2\) in the vector space i.e. \(E_1 + P_0 \approx E_2\). So, for each fact the following scoring function is computed by the Frobenius norm as follows:

\[
f(e_1, P_0, e_2) = \|E_1 + P_0 - E_2\|_F
\]  

The score is expected to be small if \(P_0(e_1, e_2)\) holds. TransE learns the entities and predicates’ embeddings in order to minimizing a reconstruction loss regarding the given facts.

Although this method is efficient and straightforward, it does not deal well with a more complex form of non-functional predicates, the predicates of which are not 1-to-1 (i.e., 1-to-N, N-to-1, and N-to-N). For example, the predicate \text{nationality}(x, y)\) translates multiple persons (as the first argument) to a unique country (as the second argument). To address this issue, many methods have been proposed such as [24, 40, 41]. In the improved methods, first, the entities’ embeddings are transferred to the predicate space then use the predicates’ embeddings to translate the transferred subject entity to the transferred object entity.

### 2.1.3 Hybrid Methods

There are some hybrid methods which use embedding representation for learning logical rules from KGs. To do so, they deploy the predicates’ embeddings in a procedure which simulates the logical inference. Thus, they can find the potential rules without exhaustive search in the space of all possible rules. Consider a logical rule \(P_1 \land P_2 \rightarrow P_3\). To imitate logical rule inferences, they define a composition function \(\circ\) in the predicates’ embeddings space which simulates logical conjunction. By assuming the left side of the rule \((P_1 \circ P_2)\) is similar to the right side \((P_3)\), they can simulate the mentioned rule in the predicates’ embeddings space. Various
kinds of composition functions are investigated such as multiplication, additive [26], and recurrent neural network [44].

All these hybrid methods suffer from lack of scalability in comparison with rule miners such as AMIE+ [8]. As a result, existing hybrid methods cannot handle real-world data such as YAGO2 and WikiData. In the following, we review two existing hybrid KG modeling systems.

EmbedRule [26] extracts CP rules from KGs by exploring in the space of predicates’ embeddings. In [26], they investigate various kinds of predicate embeddings (i.e., vector and matrix) and multiple kinds of composition functions (i.e., addition and multiplication). They show that the matrix embedding and multiplication operator setting results in best performance regarding extracting the logical rules.

Although the EmbedRule shows superior performance in the experiments [26] compared to AMIE [31], the size of benchmarks which they consider is relatively small in comparison with the benchmarks which rule miners consider (e.g. YAGO, Wikidata). The other drawback of EmbedRule is that it completely ignores the entities’ embeddings whereas these embeddings contain essential information.

Neural Logic Programming framework [14] is intended to learn probabilistic logical rules from KGs and uses CP rules as the language bias. Learning probabilistic logic model includes two aspects, learning logical structures and learning probabilistic parameters. Neural LP combines the structure and parameter learning by using TensorLog [45] system. TensorLog is an efficient implementation for answering queries submitted to the probabilistic logical knowledge base because it compiles the logical queries as differentiable functions in a neural network platform such as Tensorflow [45].

Although the Neural LP learns the first-order rules as the structure of its model, the model which it uses for predicting the missing facts is a probabilistic logical model. Consequently, it learns the probabilistic parameters of the model by maximizing the fitness of the model with the given data incrementally. This process
is rather time-consuming and restricts the space of search for the structures (first-
order rules). Due to these issues, Neural LP cannot scale over the rule miners’
benchmarks, such as YAGO2 and WikiData.

2.2 Learning Logical Models from KG Streams

In this section first, we define the KG streams and temporal rules. Second, we
review a few existing works on reasoning and learning over KG streams.

2.2.1 KG Streams and Temporal Rules

Recall that a KG consists of a set of RDF triples \((e, P, e')\), each of which expresses
a fact that entity \(e\) is related to another entity \(e'\) via relation \(P\). Such a KG is
static in the sense that temporal information is not taken into account. In this
thesis, we consider streams of KGS. A KG stream consists of a (possibly infinite)
set of quadruples of the form \((e, P, e', k)\), each of which expresses an event that
the relation \(P\) associates entity \(e\) to entity \(e'\) at time point \(k\). For convenience,
such a fact with the time stamp is also called event.

Following the convention in knowledge representation, we denote such an event as
\(P(e, e', k)\), where \(P\) is a ternary predicate, \(e\) and \(e'\) are entities, and \(k\) is a time
point constant.

Consider an KG stream \(S\) and two integers \(i, j\) with \(0 \leq i \leq j\), the \([i, j]\)-segment
\(S[i, j]\) of \(S\) is the subset of \(S\) consisting of all the events with time points between
and including \(i\) and \(j\). That is, \(S[i, j] = \{P(e, e', k) \mid P(e, e', k) \in S, i \leq k \leq j\}\).
When \(i = j\), it can be simplified as \(S[i]\). Note that \(S\) can be seen as a sequence of
KGS \(S[0], S[1], \ldots\). Sometimes we want to consider the facts via omitting the time
points in the events, and \(S^*\) denotes the static KG obtained from \(S\) by replacing
each event \(P(e, e', k)\) with the fact \(P(e, e')\).

We consider temporal CP rules (or simply temporal rules) \(\gamma\) of the following form:
Here $t$ is a time point variable and $k$ is an integer. The rule reads that if the rule body holds at time point $t$, then the rule head holds at time point $t + k$. Obviously, the class of temporal CP rules could be more general; for instance, different time points could be allowed for different atoms in the rule. Our definition of temporal CP rules is a balance of expressive power and efficiency of rule mining algorithms. Especially, in learning rules from data streams, we are interested in making predictions based on the current data stream.

\[ P_1(x, z_1, t) \land P_2(z_1, z_2, t) \land ... \land P_n(z_{n-1}, y, t) \rightarrow P(x, y, t + k). \]  

(2.12)

2.2.2 KG Streams Reasoning and Learning

The KG stream management systems comprises two folds, stream reasoning and stream learning. The stream reasoning methods are logical-based approaches which provide a framework to formalize the regulators, such as temporal rules, which are held in the data stream. These rules present the evolutionary behavior of events in the data stream. They can be applied to the previously given data to predict the upcoming events.

While there are many attempts to address stream reasoning [10], to the best of our knowledge there is no attempts to date for specified KG stream reasoning. However, there are some papers which consider temporal KGs [46, 47]. Temporal Knowledge Graph (TKG) is a directed graph in which each edge (fact) is augmented with a time stamp. It is different from the KG stream as in the stream case the system perceives data continuously while in the TKG we have entire timestamped data in the first place.
In [46], they propose a generalized notion of the property graphs, multi-attributed relational structure (MARS), to present property-sensitive rules such as the following rule:

\[
\forall x, y, z_1, z_2, z_3 : \text{spouse}(x, y)@\{\text{start} : z_1, \text{loc} : z_2, \text{end} : z_3\} \rightarrow \text{spouse}(y, x)@\{\text{start} : z_1, \text{loc} : z_2, \text{end} : z_3\}
\]

This rule necessitates that if a person \( x \) is spouse of another person \( y \) from time \( z_1 \) to time \( z_2 \) in the location \( Z_3 \), the person \( y \) is spouse of \( x \) from time \( z_1 \) to time \( z_2 \) in the location \( Z_3 \).

These papers introduce several formalisms to express the knowledge about TKG; however, they did not investigate how to extract such rules from TKG.

The stream learning systems learn a time-sensitive model from the stream of data. Again no system is proposed to address KG stream learning while a few embedding-based methods [9, 48] have been introduced to handle the learning task in the temporal KGs.

Know-Evolve [9] deploys an embedding-base approach to handle modeling the temporal KG task. To do so, it continuously learns entities’ embeddings through data with a range of timestamps while the predicates’ embeddings remain constant regarding the time. Although Know-Evolve handles the temporal KG, it does not take the stream leaning scenario into account. That is, it assumes the data regarding all time points is available to the system.

### 2.3 Transfer Learning for Knowledge Bases

A basic assumption of learning algorithms is that training and testing data belong to the same domain; consequently, a learner uses a subset of the available data (training data) to be trained; then the performance of the system is evaluated using the remaining part of data (testing data). Nevertheless, in some real-world
learning scenarios, this assumption does not hold. For example, in the cases that sufficient training data is expensive or impossible to collect. Consequently, the learner needs to deploy knowledge from other available domains as training data. This approach is called transfer learning [12, 49].

To address the logical model learning, some methods deploy transfer the learning technique [50–54]. These methods utilize Markov Logic Network (MLN) as their output formalism and KB as their input. Their input is different from KGs in two aspects, (i) the facts have arbitrary arity whereas KGs contain binary facts and (ii) their input consists of negative and positive facts whereas KG consists of merely positive facts.

The transfer learning for KBs can be defined as follows:

**Given:** A target domain with a set of facts $F_t$, a source domain with a set of facts $F_s$, where $F_s \neq F_t$ and the predicates of two domains are also different (deep transfer learning). There is also a (learned) model $R_s$ (e.g. a set of rules) over source domain.

**Learning outcome:** A model, $R_t$ over target domain.

These methods can be categorized into two folds. In the first set of methods [50, 51] the second-order logic is employed to transfer structural regularities between seemingly distinct domains. The main prior assumption in these methods is that these domains possibly share a common structure that can be extracted and expressed in the form of higher-order logic. In the second set of techniques, an explicit mapping between source and target predicates is discovered through local search methods [52–54].

One typical restrictive assumption made by all of these systems is that a pair of source and target domain is given and fixed for transfer. They do not have any mechanism to retrieve the most similar source domains from a set of learned domains. This issue may lead to a scenario in which the given source domain is not similar enough to the target domain, so the transferred knowledge harms the learning outcome. This issue is discussed extensively in the transfer learning literature as the negative transfer [12].
We are going to review TODTLER [51] and [53] which are the most recent significant systems which learn a logical model from KB through transfer learning.

Two-Order-Deep Transfer Learning system learns MLN via transferring second order rules from the source domain to the target domain. It performs the transfer learning by learning MLN clauses in the target domain according to the common patterns which are discovered in the source domain. It deploys second-order templates (SOLT) to transfer common patterns. For example, consider the notion of symmetry, which can be expressed as a second-order template, $R(x, y) \rightarrow R(y, x)$, where $R$ is a predicate variable. Hence, this template is not specific to any domain. It grounds these second-order templates in the target domain and generates a set of MLN clauses [51]. For example, the mentioned template can be instantiation by predicate $know$, $know(x, y) \rightarrow know(y, x)$.

The proposed system in [53], identifies the predicates in the target domain that are similar to the predicates in the source domain according to the information about the type of predicates in both domains. Next, it transfers the clauses (rules) from the source to the target domain and tunes the probabilistic parameters of the weighted rules by using target facts. Finally, it refines the rules by using theory refinement technique. As this system deploys type similarities to guide the search in the target domain, it does not produce domain-independent knowledge like the SOLTs which are abstract templates.

2.4 Logic and its Probabilistic Extensions

In this section, first, we review some formalisms which aim to extend logical formalisms with probabilistic features. Then we explain one of these probabilistic logical formalisms, named Logic Programs with Annotated Disjunctions (LPAD) [55]. Lastly, we explain SLIPCOVER [13], which learns LPAD from data.

In this section, all the formalisms consider more general kind of Knowledge Bases (KBs) instead of KG. These KBs contain facts with arbitrary arity, in contrast
to KGs which includes facts with binary arity. These KBs also might contain negative facts along with the positive ones. Thus, KGs is a special kind of these KBs.

### 2.4.1 Probabilistic Logical Model

The Probabilistic logic Model (PLM) is a relatively new paradigm in knowledge representation which fills the gap between logical formalisms and probabilistic models. Whereas the logical formalisms have expressive power, they cannot handle uncertainty. For presenting knowledge at the rule level the logical formalisms such as logic programs are well-established tools, but the lack of ability to handle uncertainty causes these formalisms to be less useful in some real-world problems. On the other hand, the probabilistic models (e.g. Bayesian network and Markov network) are equipped with the probability notion to manage uncertainty, whereas they do not have expressive power to represent the complex relations which are presented by logical formalisms.

To equip logical models with probabilistic features, two approaches, categorized based on semantics were developed: probabilities on interpretations and probabilities on proofs. The first group defines a probability distribution over interpretations or possible worlds (e.g. Bayesian logic program [56], Markov Logic Network [57], and Probabilistic Soft Logic [57]), while the second group defines a distribution over proofs, derivations or traces (e.g. Probabilistic Horn Abduction [58], Independent Choice Logic [59], PRISM [60], LPAD [55] and ProbLog [61]).

From another point of view, in the former approach the logical generalizations of the probabilistic graphical model were pursued, and in the latter approach probabilistic generalizations of logical models were pursued. Furthermore, some less widespread approaches were investigated, such as using second-order logic for presenting probabilistic features directly. The lack of efficient methods in reasoning and learning for these formalisms is a significant issue [62].
The second group of languages following the distribution semantics [63] diverge regarding the method by which they define the distribution over proofs, but have an equal expressive power: There are transformations with linear complexity that can convert each one into the others [61]. In the following, we review in detail Logic Programs with Annotated Disjunctions [55] from the second category. The advantages of Logic Programs with Annotated Disjunctions (LPAD) are their expressive power and simplicity.

LPAD allows disjunction in the heads of program rules to express probabilistic multiple choices [55]. Similar to other classes of (probabilistic) logic programs, the fragment of LPAD logic is based on three types of symbols: constants, variables, and predicates. Constants are names for specific objects (entities) in the domain; variables range over objects in the domain, and predicates represent relations or features among objects. We use $c$ (possibly with subscripts) or words starting with a lower-case symbol for constants; $x$, $y$ and $z$ or sometimes with subscripts for variables; $P$ or words starting with a capital symbol for predicates. Variables and constants are typed. An atom is of the form $P(t_1, \ldots, t_n)$ where $P$ is a predicate and each $t_i$ is a constant or variable ($1 \leq i \leq n$). $P(t_1, \ldots, t_n)$ is a ground atom if every term $t_i$ is a constant. By considering the type restriction, the variables and constants which are assigned to an argument of a predicate should be compatible with the type of that argument. For example, $\text{Actor}(\text{person})$ indicates the type of first argument of $\text{Actor}$ is $\text{person}$ so in atom $\text{Actor}(x)$ the type of variable $x$ should be $\text{person}$.

A probabilistic atom is a pair $(a, p)$ where $a$ is an atom and $p$ is a probability degree $(0 \leq p \leq 1)$. $(a, p)$ is to represent that the atom $a$ is true with the probability $p$. A literal is an atom or its negation.

Formally, a LPAD rule is of the form

$$(a_1 : p_1); \cdots; (a_m : p_m) \leftarrow b_1, \ldots, b_n$$
where \((a_1 : p_1), \ldots, (a_m : p_m)\) are probabilistic atoms, \(b_1, \ldots, b_n\) are literals. ‘;’ is for disjunction. Informally, the above rule reads that if \(b_1, \ldots, b_n\) are true, then at least one of \((a_1 : p_1), \ldots, (a_m : p_m)\) is true.

For example, considering the following rule about a hotel room alarm system:

\[
(Fire : 0.7); (Smoking : 0.2) \leftarrow \text{Alarm activated}(x), \text{Room occupied}.
\]

This rule imposes that if a sensor \((x)\) activates the alarm and the related room is occupied (by a guest), with high probability \((0.7)\) there is fire in the room, with low probability \((0.2)\) the guest is smoking and with \(0.1\) probability nothing is happening, and it is a faulty alarm.

The semantics of LPADs is defined by the distribution semantics, which is first introduced for PRISM [60]. In the distribution semantics for LPADs, each world is established by choosing one atom from the head of grounding form of an LPAD rule, and the probability \(p_i\) of the head atom \((a_i, p_i)\) is computed by accumulating the probability of worlds in which the model evaluates \(a_i\) as true.

We aim to learn LPADs from KBs in Chapter 6 through a novel transfer learning system.

There are two specific learners initiated for learning LPAD, SLIPCACE [64] and its improved version SLIPCOVER [13]. Both of these learners use a top-down approach to explore the space of hypotheses. They utilize log likelihood of the obtained LPAD rules on the given data as the heuristic of their search.

Typically learning PLM is distinguished by the two problems. First, there is the problem of parameter learning. The objective of this task is estimating suitable values for the parameters of a model, in which the structure is fixed. Second, there is the problem of structure learning in which the learner must deduce both the structure and the parameters of the model from data. For example, SLIPCOVER learns the structure and parameters of LPAD from data.
The input of SLIPCOVER is mega-examples. Each mega-example contains positive and negative examples for all predicates that may appear in the head of learned LPAD rules. The mega examples are distinct from each other while facts of each mega-example are connected through the shared entities. This kind of input data is different from KGs which we introduced in the previous sections due to the presence of negative facts and the arbitrary arity of facts.

SLIPCOVER identifies proper candidate rules and then searches for the best theory (set of rules). It uses the log likelihood of rules based on the data as a heuristic. SLIPCOVER deploys EMBLEM [65] to learn the parameters of LPAD rules. To do so, EMBLEM performs the Expectation Maximization (EM) over BDDs, which is an optimized method. Using heuristic search and deploying an efficient parameter learner (EMBLEM) enable SLIPCOVER to learn LPAD rules from KBs efficiently.
Chapter 3

Learning Rules via Learning Representation from KGs

In this chapter, we propose an approach to learn logical rules by representation learning, named RLvLR. It learns first-order rules from KGs and infers the new facts by applying the learned rules. RLvLR is scalable and able to handle vast KGs such as YAGO2 and Wikidata. RLvLR learns rules from KGs by using the technique of embedding in representation learning together with a new sampling method.

The experiments show that RLvLR outperforms some state-of-the-art rule learners including AMIE+. Besides, we used the learned rules in an inference procedure to handle the link prediction task. In this task, RLvLR outperformed Neural LP, a state-of-the-art link prediction system, in both runtime and accuracy.

This chapter resulted in the following publication:

3.1 Introduction

Knowledge graphs (KGs) have proven to be compelling and robust data management approach that underlies several knowledge bases such as YAGO [1], DBpedia [2], Wikidata [3] and Freebase [4].

Due to the large data volume of KGs, it is impossible to construct large KGs manually. Thus, a major task in KG construction is to develop scalable methods for automated learning of new entities, their properties and relationships. As some researchers have pointed out, a KG is not just a graph database [18]. In particular, it should have a layer of conceptual knowledge, which is usually represented as a set of rules like BornIn($x, y$) $\land$ Country($y, z$) $\rightarrow$ Nationality($x, z$), meaning that if person $x$ was born in city $y$ and $y$ is in country $z$, then $x$ is a citizen of $z$. Rules are explicit knowledge (compared to a neural network) and can provide human understandable explanations to learning results (e.g., link prediction) based on them. Thus, it is useful and important to extract rules for KGs automatically.

Traditional methods of rule learning cannot be directly employed in rule construction over KGs for several reasons. First, those methods are not scalable enough to handle the huge amount of data contained in common KGs. For example, DBpedia 3.8 has more than 11M facts, which poses a challenge to most existing methods. Moreover, KGs do not explicitly express negative examples, which are essential for many data mining tools.

In the literature, many new approaches have been proposed to learn rules of various forms from a variety of databases. The problem of mining rules via exploring the space of all possible hypothesizes (rules) has been investigated in the paradigm of Inductive Logic Programming (ILP) [66]. Some efficient rule miners for KGs have been developed lately, including SWARM [5], RDF2rules [6], ScaleKB [7] and AMIE+ [8]. They are much more efficient than their predecessors and are able to learn rules from large datasets. However, scalability is still a major challenge for existing systems.
On the other hand, in the paradigm of representation learning, statistical predictive models have been widely applied in learning facts that are absent in a KG. Tensor factorization [15] and the translation-based approach [24] are two major approaches in this category. The basic idea is to encode relational information by using low-dimensional representations (embeddings) of entities and predicates. Such representation learning techniques have been applied in learning rules in KGs [26, 44]. This is a promising research direction for rule learning in large KGs but the resulting systems are still not very efficient compared to some major rule miners such as AMIE+ [8].

In this chapter, we tackle this challenge by providing a scalable method for learning rules in KGs. The main idea is to define new embeddings (of arguments) and new scoring functions. While in previous system [26] the similarity is based on synonymity, we also consider co-occurrence to characterize similarity. In this sense, the scoring function is more precise, so the number of learned quality rules is higher. Then we use them to guide the extraction of rules and thus reduce the search space. Besides, we proposed a novel sampling method and efficient rule evaluating mechanism that allows our system to handle massive benchmarks efficiently.

We have implemented a system prototype RLvLR\textsuperscript{1} based on our new methods and compared it with the state of art system AMIE+ on major benchmarks such as YAGO, DBpedia and Wikidata for the rule mining task. Our experimental results show that RLvLR outperformed AMIE+ in both time efficiency and the number of mined quality rules. For example, in a learning task for 20 specific predicates from Wikidata (which has over 8M facts), RLvLR mined more than 56 rules on average in 2.41 hours but AMIE+ mined 1 rule on average in 10 hours (with the same rule quality thresholds). We have also implemented an inference method to predict new facts. RLvLR also outperformed Neural LP \textsuperscript{[14]} on link prediction regarding scalability and accuracy.

\textsuperscript{1}https://www.ict.griffith.edu.au/aist/RLvLR/
3.2 An Overview of Our Approach

In this section, we present our embedding-based approach to rule learning in a nutshell. We focus on discriminative rule mining, that is, specifying a target predicate $P_t$ in a KG, to mine quality rules whose head has the predicate $P_t$.

In contrast to ILP approaches, instead of using a refinement operator to search the rule space, we use embedding models to effectively prune the search. However, this is not straightforward. In order to develop an efficient algorithm for rule learning problem using embedding models, we need to resolve two issues. First, existing embedding models do not work directly for vast KGs. For instance, RESCAL is unable to handle YAGO2 [15]. We address this issue by introducing a new sampling algorithm to restrict the range of entities to be considered so that embeddings are computed only for those entities that are relevant to the target predicate. Moreover, we need a more efficient and refined approach to rank candidate rules. To this purpose, we first define the embeddings of arguments of predicates and then introduce scoring functions that allow fast ranking of rules, based on embeddings of entities, predicates and arguments (instead of only entities and predicates). In addition, efficient numerical algorithms for matrices are also employed to speed up our method.

The above two aspects are also major sources where the efficiency of RLvLR is achieved. Our method for rule learning is summarised in the following algorithm, while some major components in the algorithm will be explained later.

In Algorithm 1, the integer $len$ is for the maximum length of rules to be learned; $MinSC$ and $MinHC$ set the minimum values of standard confidence and head coverage for learned rules, respectively. Due to the vast size of the input KG $\mathcal{K} = (E,F)$, it is necessary to sample the data first. For this purpose, we use a sampling method $\text{Sampling}()$ to obtain an (often much) smaller KG $\mathcal{K}' = (E',F')$ that contains only those entities and facts that are relevant to the target predicate $P_t$. To learn rules of maximum body length $len = n$, we generate sample entities in an incremental manner $E_0, \ldots, E_{n-1}$ as follows:
Algorithm 1 Learn rules for a KG and a target predicate

Input: a KG $\mathcal{K}$, a predicate $P_t$, an integer $\text{len} \geq 2$, and two real numbers $\text{MinSC}, \text{MinHC} \in [0,1]$

Output: a set $\text{Rule}$ of CP rules

1: $\mathcal{K}' := \text{Sampling}(\mathcal{K}, P_t, \text{len})$
2: $(\mathcal{P}, \mathcal{A}) := \text{Embeddings}(\mathcal{K}')$
3: $\text{Candidates} := \emptyset$
4: for $2 \leq l \leq \text{len}$ do
5: \hspace{1em} $\text{Add RuleSearch}(\mathcal{K}', P_t, \mathcal{P}, \mathcal{A}, l)$ to $\text{Candidates}$
6: end for
7: $\text{Rules} := \text{Evaluate}(\text{Candidates}, \mathcal{K})$
8: $\text{Rules} := \text{Filter}(\text{Candidates}, \text{MinSC}, \text{MinHC})$
9: return $\text{Rules}$

- $E_0$ consists of the entities that are connected to another entity in $E$ by $P_t$, i.e., $E_0 = \{e \mid$ there exists an entity $e'$ in $E$ s.t. $P_t(e, e') \in F$ or $P_t(e', e) \in F\}$;
- $E_i$ ($0 < i < n$) consists of the entities that are connected to another entity in $E_{i-1}$ by any predicate $P$, i.e., $E_i = \{e \mid$ there exists an entity $e'$ in $E_{i-1}$ s.t. $P(e, e') \in F$ or $P(e', e) \in F\}$;

Since we are interested only in CP rules with no more than $n$ body atoms, the subset $E' = \bigcup_{i=0}^{n-1} E_i$ of all the entities in $\mathcal{K}$ covers almost all information needed for mining such rules. We thus generate a sample set $F'$ of facts as follows:

$$F' = \{P(e_1, e_2) \mid e_1, e_2 \in E', P(e_1, e_2) \in F\}. \tag{3.1}$$

After sampling, the next step, i.e., $\text{Embeddings()}$, is to obtain the embeddings $\mathcal{P}, \mathcal{A}$ for respectively predicates and arguments in $\mathcal{K}'$. Existing systems such as RESCAL usually compute embeddings only for entities and predicates. We will define the notion of argument embeddings later based on entity embeddings, which together with embeddings of predicates provide more refined scoring functions for candidate rules.

The search for candidate CP rules of the form $P_1(x, z_1) \land P_2(z_1, z_2) \land \ldots \land P_n(z_{n-1}, y) \rightarrow P_t(x, y)$ is actually reduced to the search for plausible paths, that is, sequences of predicates $P_1, P_2, \ldots, P_n$ and their inverses. This is achieved through the method
RuleSearch(). We use the proposed scoring function to guide and prune the search, which turns out to be rather effective for extracting rules. Then the selected candidates are kept for the final evaluation.

Finally, the candidates are evaluated according to their SC and HC before being returned. This is achieved through the method Evaluate(). To compute the SC and HC of candidate rules, we utilise the efficient matrix multiplication. The embedding of arguments, the new scoring function and the employment of matrix algorithms greatly contributed to the scalability of our approach. More technical details will be explained in Sections 3.3 and 3.4.

To show the usefulness of extracted rules, we implemented an inference method that predicts new facts with a degree of confidence based on the given facts and first-order rules that are augmented with SC scores. To obtain the confidence degree (CD) of a fact, we adapt the \( score^*(\cdot) \) function from [8] by aggregating the SC of all the rules inferring the facts in a Noisy-OR manner. The intuition is that facts inferred by more rules should have a higher confidence degree. Instead of using the PCA scores as in [8], we use SC as it is easier to compute. Formally, for a fact \( f = P(e, e') \) and the set of rules \( R \) that can infer \( f \) from the given KG, the CD of \( f \) is defined as follows:

\[
CD(f) = 1 - \prod_{r \in R} (1 - SC(r))
\]

With this module, RLvLR becomes an end-to-end learner that is able to handle the link prediction task.

### 3.3 Scoring Functions

As explained above, the task of searching for CP rules can be reduced to that of searching for plausible paths of predicates. This is done by introducing scoring functions over all possible paths. To define such a scoring function, note that a path \( p = P_1, P_2, \ldots, P_n \) can be seen as a binary predicate between the starting
entity and the ending entity, and $p$ is plausible if the pairs of entities associated
by the path are similar to those associated by the target predicate $P_t$. Such a
similarity between $p$ and $P_t$ is referred to as synonymy, where two predicates
associate similar pairs of entities. For instance, the predicates $\text{BornIn}(u, v)$ and
$\text{LiveIn}(u, v)$ associate similar pairs of a person $u$ and a place $v$.

Based on this intuition, a scoring function is defined using an embedding model
in [26]. We use the following rule $r$ of length 3 to illustrate the method:

$$P_1(x, z) \land P_2(z, y) \rightarrow P_t(x, y).$$

The embedding of path $p = (P_1, P_2)$ is $P_1 \cdot P_2$ (multiplication of $P_1$ and $P_2$),
which should be similar to the embedding of $P_t$, denoted $P_1 \cdot P_2 \approx P_t$. Hence, the
synonymy scoring function is

$$f_1(r) = \text{sim}(P_1 \cdot P_2, P_t)$$  \hspace{1cm} (3.2)

where $\text{sim}$ is defined by the Frobenius norm as follows, for two matrices $M_1$ and
$M_2$,

$$\text{sim}(M_1, M_2) = \exp(-\|M_1 - M_2\|_F).$$

Yet for rules gets longer than 2, the computation of synonymy scoring function
involves nesting of matrix manipulation and become less effective for exploring the
search space as it involves whole paths. Thus, we propose a local scoring function
based on co-occurrence. Besides synonymy, co-occurrence is also widely studied
in natural language processing [67]. In our context, $P_1$ and $P_2$ are adjacent in a
path only if they share many entities in the KG. For example, $\text{LiveIn}(u, v)$ and
$\text{LocatedIn}(v, w)$ share cities $v$.

To define a co-occurrence scoring function, we first introduce the notion of argument
embeddings. For an argument of a predicate, its embedding is defined as
the average value of the embeddings of all the entities appearing in the position
of this argument. Formally, the embeddings of the subject and object argument
of a predicate $P$ are defined as:

$$P^s = \frac{1}{n} \sum_{e \in E_p^s} k_e.E \quad \text{and} \quad P^o = \frac{1}{n} \sum_{e \in E_p^o} l_e.E$$

where $n = \# \{P(e, e') \in F'\}$, $E_p^s = \{e \mid \exists e' \text{ s.t. } P(e, e') \in F'\}$, $E_p^o = \{e' \mid \exists e \text{ s.t. } P(e, e') \in F'\}$, $F'$ is defined by Eq.(3.1), $k_e = \# \{P(e, e') \mid \exists e' \text{ s.t. } P(e, e') \in F'\}$ and $l_e = \# \{P(e', e) \mid \exists e \text{ s.t. } P(e, e') \in F'\}$. In path $p$, the embedding of the object argument of $P_1$ should be similar to the embedding of the subject argument of $P_2$, denoted $P_1^o \approx P_2^s$. Hence, the co-occurrence scoring function is

$$f_{loc}(P_1, P_2) = \text{sim}(P_1^o, P_2^s)$$

Similarly, we should have $P_1^s \approx P_1^s$ and $P_2^o \approx P_2^t$, as well as the corresponding co-occurrence scoring functions

$$f_{loc}(P_1, P_t) = \text{sim}(P_1^s, P_t^s), f_{loc}(P_2, P_t) = \text{sim}(P_2^o, P_t^o)$$

We aggregate the local scoring functions as follows.

$$f_2(r) = f_{loc}(P_1, P_2) + f_{loc}(P_1, P_t) + f_{loc}(P_2, P_t). \quad (3.3)$$

In Figure 3.1 we illustrate the corresponding paths for the mentioned rule. While the scoring function $f_1$ considers the predicates in the paths, with $f_2$, we consider the shared variables that are involved in the paths. Consequently, we use both of these two scoring functions, which complement each other.

### 3.4 Rule Evaluation

In the \texttt{Evaluate}() method, for efficiency, we first evaluate the picked candidate rules based on the sampled KG $\mathcal{K}'$, which provides us an estimated quality of rules. We return the set of rules with $supp \geq 1$. These rules may still contain a large number
of redundant and low quality rules and thus it is necessary to further clean the returned rules based on two measures SC and HC.

To compute SC and HC degrees, a method is required to check the satisfiability of body atoms of all candidate rules in the last phase. In other words, we need to figure out all relevant atoms that can trigger a candidate rule. In this task, we have a KG (e.g. \( \mathcal{K} \) or \( \mathcal{K}' \)) and a CP rule as input. Let \( E = \{e_1, \ldots, e_n\} \) be the set of all entities and \( \mathcal{P} = \{P_1, \ldots, P_m\} \) be the set of all predicates. We can represent the input KG as a set of \( S \) matrices like RESCAL, where each \( n \times n \) matrix \( S(P_k) \) corresponds to a predicate \( P_k \) in the KG \( (1 \leq k \leq m) \). Specifically, the \((i, j)\) entry \( S(P_k)[ij] \) of the adjacency matrix \( S(P_k) \) is 1 if the fact \( P_k(e_i, e_j) \) is in the input KG; 0 otherwise. Thus, \( S(P_k) \) is a binary one.

There is a close connection between the product of adjacency matrices for predicates and the closed-path rules. We use an example to explain the idea. Consider the rule \( r: P_1(x, z) \land P_2(z, y) \rightarrow P_t(x, y) \). We say a fact \( P_t(e, e') \) is inferred by the rule \( r \) in a KG \( \mathcal{K} = (E, F) \) if \( P_1(e, e'') \in F \) and \( P_2(e'', e') \in F \) for some \( e'' \in E \). Then the product \( S(P_1) \cdot S(P_2) \) of two matrices \( S(P_1) \) and \( S(P_2) \) is the adjacency matrix of the set of facts that are inferred by \( r \).

The \([i, j] \) entry of \( S(P_1) \cdot S(P_2) \) represents the number of rule paths that start from \( e_i \), traverse via \( P_1 \) to another entity and so on but finally go to \( e_j \) via \( P_2 \). From a matrix \( S \), we can obtain a binary matrix \( \text{binary}(S) \) by setting all non-zero entries of \( S \) as 1. Let \( S(P_1, P_2) = \text{binary}(S(P_1) \cdot S(P_2)) \). Note that although we explain
the case where rule length is three, it is straightforward to generalize this relation to rules of any length.

Let $r$ be $P_1(x, z) \land P_2(z, y) \rightarrow P_t(x, y)$ and $\mathcal{K} = (E, F)$ where $E = \{e_1, e_2, e_3\}$ and $\mathcal{K} = \{P_1(e_1, e_2), P_1(e_2, e_1), P_1(e_1, e_3), P_2(e_2, e_3), P_2(e_2, e_1), P_2(e_3, e_3), P_t(e_1, e_3)\}$.

The matrices for the predicates $P_1$, $P_2$ and $P_t$ are:

$S(P_1) : \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, $S(P_2) : \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, $S(P_t) : \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Then

$S(P_1) \cdot S(P_2) = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ binary(·) $\rightarrow S(P_1, P_2) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

The last matrix represents the pairs that are inferred by the rule: $\{(e_1, e_1), (e_1, e_3)\}$. Since we have just one fact for the target predicate, $P_t(e_1, e_3)$, the quality degrees of rule $r$ are $HC = 1$ and $SC = 0.5$, respectively.

### 3.5 Experiments

Based on the methods presented in previous sections, we have implemented a system RLvLR (Rule Learner via Learning Representation) and conducted two sets of experiments to evaluate the new system. The executable codes, benchmark datasets and experimental results are publicly available at [https://www.ict.griffith.edu.au/aist/RLvLR/](https://www.ict.griffith.edu.au/aist/RLvLR/).

The first set aims to evaluate the scalability of RLvLR and the number of quality rules learned by the system, while the second set is to evaluate the scalability and accuracy of RLvLR for link prediction. The benchmark datasets adopted
in our experiments include various versions of Freebase, YAGO, DBpedia and Wikidata that are widely used in experimental evaluations by major systems for rule learning and link prediction in KGs. RLvLR was compared to state-of-the-art rule miners such as AMIE+ [8] and ScaleKB [7], as well as state-of-the-art system for link prediction, Neural LP [14]. Our experiments were designed to validate the following statements:

1. RLvLR is much faster than major rule mining systems such as AMIE+ for large-scale KGs.

2. RLvLR is able to mine significantly more quality rules than AIME+, especially on vast datasets such as DBpedia 3.8 and Wikidata.

3. Regarding link prediction, RLvLR outperforms the state-of-the-art link prediction system Neural LP in terms of both scalability and accuracy.

The five benchmark datasets are specified in Table 3.1, where the last three have been often used in rule mining [7, 8]. FB15K-237 [68] (aka. FB15KSelected) and FB75K (from NIPS’13 dataset) are obtained from Freebase and widely adopted for link prediction benchmarking [14].

<table>
<thead>
<tr>
<th>KG</th>
<th># Facts</th>
<th># Entities</th>
<th># Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB15K-237</td>
<td>310K</td>
<td>15k</td>
<td>237</td>
</tr>
<tr>
<td>FB75K</td>
<td>316K</td>
<td>75k</td>
<td>13</td>
</tr>
<tr>
<td>YAGO2s</td>
<td>4.12M</td>
<td>1.65M</td>
<td>37</td>
</tr>
<tr>
<td>Wikidata</td>
<td>8.40M</td>
<td>4.00M</td>
<td>430</td>
</tr>
<tr>
<td>DBpedia 3.8</td>
<td>11.02M</td>
<td>2.20M</td>
<td>650</td>
</tr>
</tbody>
</table>

Table 3.1: Benchmark specifications

For benchmark KGs FB15K-237 and FB75K, we used a PC with Intel Core i5-4590 CPU at 3.30GHz × 4 and with 5GB of RAM, running Ubuntu 14.04. For other larger benchmark KGs we tested, the experiments were conducted on a server with Intel Xeon CPU at 2.67GHz (one thread) and with 40GB of RAM, running RedHat Linux 6.1.
3.5.1 Rule Mining

This set of experiments concerns mining quality rules. The rule quality was measured by standard confidence (SC) and head coverage (HC) [8]. Note that while we used sampling for rule mining, HC and SC degrees of mined rules are computed over the whole datasets (i.e., not on the samples).

Experiment 1 We randomly selected 20 target predicates for YAGO2, Wikidata and DBPedia. A 10 hour limit was set for each target predicate. Table 3.2 shows the average numbers of quality rules (#R, SC $\geq 0.1$ and HC $\geq 0.01$ as in [8]) and numbers of high quality rules (#QR, SC $\geq 0.7$) mined for selected target predicates and the running times (in hours, averaged over the targets) of RLvLR and AMIE+.

Compared to AMIE+, RLvLR showed better performance in terms of both the runtime and the numbers of mined quality rules. The superiority of RLvLR is more obvious in mining high quality rules. Note that RLvLR deployed the same redundancy elimination as AMIE+, and the numbers were obtained after the redundancy elimination.

To estimate the predictive power of the corpus of mined rules, we eliminated from each benchmark 30% of its facts (up to 5K facts) involving the target predicates and checked how many facts (including the eliminated ones) can be predicated by applying mined rules on the remaining facts. Table 3.3 shows the numbers of predicted facts (# Facts) and those predictions with CD $\geq 0.9$ (#QFacts). In this part, we consider five target predicates in all three benchmarks. Note that while AMIE+ mined some non-CP rules whose application cannot be implemented using

<table>
<thead>
<tr>
<th>KG</th>
<th>RLvLR</th>
<th>AMIE+</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#R</td>
<td>#QR</td>
</tr>
<tr>
<td>YAGO2s</td>
<td>6.3</td>
<td>1.8</td>
</tr>
<tr>
<td>DBpedia 3.8</td>
<td>42.7</td>
<td>9.2</td>
</tr>
<tr>
<td>Wikidata</td>
<td>56.8</td>
<td>25.6</td>
</tr>
</tbody>
</table>

Table 3.2: Rule mining comparison between RLvLR and AMIE+
our inference method, the majority of them can be applied: 77% for YAGO2s, 100% for DBpedia, and 100% for Wikidata.

The numbers of quality rules mined by RLvLR on YAGO2s, DBpedia 3.8 and Wikidata are 1.1, 4.7 and 59.7 times of those mined by AMIE+ (from Table 3.2), where as the facts predicated by RLvLR are 4.1, 10.3 and 12.3 times of those predicated by AMIE+. This suggests the usefulness of the additional rules mined by RLvLR in link prediction.

Since we were unable to run ScaleKB system or obtain the Freebase benchmark used in [7], we could only compare the rules mined by RLvLR on YAGO2s with those reported in [7]. As they only reported rules with length up to 3, we restricted ourselves to rules of length 3 too. For instance, we observed that the following rules that were learned by RLvLR but not by ScaleKB. The two numbers preceding a rule denote SC and HC degrees of the rule, respectively.

\[
0.82, 1 : \text{isAffiliatedTo}(x, y) \rightarrow \text{playsFor}(x, y).
\]

\[
1, 0.82 : \text{playsFor}(x, y) \rightarrow \text{isAffiliatedTo}(x, y).
\]

We also observed a large number of informative rules of length 4 learned by RLvLR but neither ScaleKB nor AMIE+ could learn them. For example, the following pattern with three body atoms:

\[
0.89, 0.13 : \text{hasChild}(x, t) \land \text{hasChild}^{-1}(t, z) \land \\
\text{isCitizenOf}(z, y) \rightarrow \text{isCitizenOf}(x, y).
\]
Table 3.4: Number of mined rules with different scoring functions

<table>
<thead>
<tr>
<th>Target Predicate</th>
<th>RLvLR</th>
<th>RLvLR*</th>
</tr>
</thead>
<tbody>
<tr>
<td>formOfGovernment</td>
<td>137</td>
<td>97</td>
</tr>
<tr>
<td>awardWinner</td>
<td>1024</td>
<td>574</td>
</tr>
<tr>
<td>parentGenre</td>
<td>52</td>
<td>30</td>
</tr>
<tr>
<td>sameDirector</td>
<td>691</td>
<td>284</td>
</tr>
<tr>
<td>eventLocation</td>
<td>229</td>
<td>141</td>
</tr>
</tbody>
</table>

This rule means that if a person $x$ has a common child $t$ with somebody else $z$ and the person $z$ is a citizen of some place $y$, then the first person $x$ is also the citizen of place $y$.

Experiment 2  We have conducted an experiment to demonstrate that our new scoring function is more informative than the scoring function defined in [26]. For this purpose, we implemented a system, named RLvLR*, by replacing the scoring function in RLvLR with the scoring function in [26]. We set a 5 hours time limit, $SC \geq 0.01$ and $HC \geq 0.001$. As FB15k-401 KG was used in [26], our experiment was conducted on the similar benchmark, FB15K-237. The experimental results are summarised in Table 3.4, which show that our scoring function was capable to mine (up to 2.4 times) more rules than that reported in [26].

3.5.2 Link Prediction

The second set of experiments aim to evaluate the predictive power of mined rules for link prediction in KGs. Specifically, our experiments show that, for the task of link prediction, RLvLR significantly outperforms Neural LP in terms of scalability, while the accuracy of RLvLR is comparable to that of other systems for link prediction. So, the major advantage of RLvLR is in its capability of handling vast KGs with a accuracy that is comparable to other major systems. Note that our goal is not to compete with them on accuracy of link prediction for relatively small KGs.
Given a KG, the task of link prediction is to identify for each predicate $P$ and each entity $e$, an entity $e'$ such that $P(e, e')$ is in the KG; or alternatively, to identify for each predicate $P$ and each entity $e'$, an entity $e$ such that $P(e, e')$ is in the KG.

We conducted two experiments for link prediction. The first one is to demonstrate the scalability of RLvLR while the second one is to show that RLvLR is comparable to major systems of link prediction in terms of accuracy.

Following the experiments of Neural LP [14], we used two metrics Mean Reciprocal Rank (MRR) and Hits@10. MRR is the average of the reciprocal ranks of the desired entities and Hits@10 is the percentage of desired entities being ranked among top ten.

**Experiment 1** In this experiment, we compared RLvLR with Neural LP [14] on two benchmark datasets FB75K and WikiData. The two datasets FB75K and WikiData have 75K and 4M entities, respectively. Each dataset is divided into training set (70%) and test set (30%). Neural LP is a state-of-the-art system for link prediction based on rule learning and it features for its scalability. It was able to handle FB75K while it could not handle Wikidata in our experiment. We note that the largest dataset for comparing Neural LP with other systems is FB15K-237, which is still much smaller than FB75K regarding the number of entities. We ran both RLvLR and Neural LP on these two datasets for link prediction. While all 13 predicates of FB75K were tested, 50 randomly selected predicates for Wikidata were tested as it has too many predicates (430). The experimental results are summarised in Table 3.5.

The parameters of RLvLR were set to $SC \geq 0.005$ and $HC \geq 0.001$ to achieve better accuracy. The time unit was hour and a 5-hour time limit was set for each target predicate.

**Experiment 2** In the literature, three benchmarks WN18, FB15K and FB15K-237 are usually used for evaluating link prediction [14, 15, 24]. As pointed out in [14], the challenging benchmark for state-of-the-art systems is FB15K-237 as the test entities are rarely directly linked in the KG, a system for link prediction
needs to reason explicitly about compositions of relations. The CP rules learned by RLvLR and Neural LP can naturally capture such relation compositions. So, we selected FB15K-237 as the benchmark for our experiment.

Yang et al. selected five major systems for comparing with their Neural LP but for the dataset FB15K-237, experimental results are only available for the two systems DISTMULT and Node+LinkFeat. As a result, we included these two systems and Neural LP for comparison. The test results for these three systems are extracted from the experimental results for Neural LP [14]. We compare the performance of RLvLR with the three systems and the experimental results are summarized in Table 3.6.

From the results, RLvLR obtained better Hits@10 than Neural LP and Node+LinkFeat while DISTMULT is slightly better. Regarding the MRR measure, all of these systems performed quite similar.

### 3.6 Conclusion

In this chapter, we have proposed a system RLvLR for extracting closed-path rules, a special but useful class of first-order rules, from RDF Knowledge Graphs (KGs). RLvLR mines in the space of closed-path rules (hypothesizes) by exploring...
the space of embeddings of predicates and arguments. This is achieved by a novel embedding model and new scoring functions. A target oriented sampling has also been proposed, which significantly contributes to the scalability of RLvLR in handling large KGs with over 10 million facts. In a rule mining system, a challenging and time-consuming task is about how to evaluate candidate rules, and we reduce its computation to a series of matrix operations.

Our experimental results demonstrate that RLvLR outperforms AMIE+ in terms of rule quality and efficiency. For link prediction, RLvLR outperforms Neural LP in terms of efficiency and accuracy.
Chapter 4

Learning Temporal Rules from KG Streams

In the previous chapter, we proposed a learning approach which extracts first-order rules from a static KG. However, in many real-world problems, we need to consider a stream of facts. In this case, each fact is augmented with a time stamp which indicates the happening time of the fact (event).

In this chapter, we attempt to address the problem by presenting an approach to learn temporal rules from KG streams. The learned temporal rules are applied in link prediction (event prediction) over KG streams. Our proposed method, StreamLearner, uses the RLvLR to obtain the structure of temporal rules. It also deploys the extended version of rule quality measures to assess the learned temporal rules. Our experimental results show that StreamLearner is effective and efficient in learning temporal rules on real-world datasets and significantly outperforms some state-of-the-art static link prediction systems including RLvLR, HOLE and TransE.

This chapter resulted in the following work:

Pouya Ghiasnezhad Omran, Kewen Wang, Zhe Wang: Learning Temporal Rules from Knowledge Graph Streams. In the AAAI Spring Symposium Series 2019 [accepted for publication]
4.1 Introduction

An increasing number of KGs are emerging that model events taking places over time besides static relations among entities. However, the static rule learning systems like RLvLR customarily assume KGs to be static and are not suitable for KGs with constantly evolving data. A KG containing constantly evolving data can be viewed as a stream of snapshots of the KG over a sequence of time points. Figure 4.1 illustrates such a KG stream, which involve four entities and some events occurred among them at various time points. Entities $e_0$, $e_1$, and $e_2$ are three countries, and $e_3$ is the missile. Three events occurred in the past: Country $e_0$ test missile at time point $\tau - 4$, and then it established military cooperation with country $e_1$ and imposed sanction on country $e_2$ at time point $\tau-3$. Besides classical link prediction questions such as “Which countries besides $e_0$ tested missile?” or link prediction questions with temporal knowledge such as “Which counties may attack $e_2$ at the time point $\tau$?” , there are also schema level questions that of much interest, such as “After a country $x$ tested missile and then imposed sanction on another country $y$ on the next day, in how many days is it most likely that $x$ will attack $y$?”

This brings in some research challenges. First, this requires temporal knowledge to be expressed in the schema information. For example, the above question can be expressed as a temporal rule $\text{test}(x, \text{missile}, t) \land \text{imposeSanc}(x, y, t + 1) \rightarrow \text{attack}(x, y, t + k)$, where $k$ ranges over integers. Learning a temporal rule with

![Figure 4.1: Example of KG stream.](image)
a specific $k$ and a highest confidence degree indicates an answer to the above question. In fact, our experiments show that using temporal rules also improves the accuracy of classical link prediction over KG Streams (see Section 4.4). Moreover, existing rule learners for KGs cannot be directly used to learn temporal rules. Finally, a framework for stream learning and reasoning of temporal rules is still missing.

In this chapter, we make the first attempt to tackle this problem by developing a method for learning temporal rules from KG streams. In our proposed method, an existing rule learner for KG is used to extract static rules from an initial data set, which is the collection of facts at the first few time points. These static rules, referred to as structure rules, are used to construct a space of candidate temporal rules. Then we extract temporal rules from this rule space by generalising major quality measures for static rules to temporal rules. Based on this method, we have implemented a system called StreamLearner. Experiments have been conducted to evaluate the proposed method. Our experiments show that StreamLearner outperforms the state-of-the-art static KG learners including HOLE [15] and TransE [16] regarding the link prediction in the streaming scenario. We also show that (temporal) rule learning from KG data streams and the corresponding link prediction can be performed effectively and efficient over various configurations of the stream. For example, it can forecast the events for different time points ahead with acceptable accuracy.

4.2 Temporal Rule Learning

Existing rule learning methods generally involve two components: one is to learn the structure of the rules (e.g., through search and refinement) and the other is to filter out low quality rules through certain quality measures. Our temporal rule learning also involve such two components.

Consider a KG stream $S$, to learn the structure of the rules, our method uses facts in an initial segment of the stream $S^*[0,n]$ ($n \geq 0$), which consists of the facts
from time points 0 up to \( n \), as *structure training data*. A static rule learner such as RLvLR (chapter 3) can be used to learn a set of CP rules \( R \). Such a CP rule is referred to as a *structure rule*. That is, a structure rule is a static rule learned from the structure training data.

Given CP rule \( r \) of the form \( P_1(x, z_1) \land P_2(z_1, z_2) \land \ldots \land P_n(z_{n-1}, y) \rightarrow P(x, y) \)

for each integer \( k \geq 0 \), we obtain a temporal rule \( r^{(k)} \) of the following form:

\[
P_1(x, z_1, t) \land P_2(z_1, z_2, t) \land \ldots \land P_n(z_{n-1}, y, t) \rightarrow P(x, y, t + k).
\]

Then to assess the quality of candidate temporal rules, we adapt standard measures, namely support, standard confidence, and head coverage, that are used in rule learning literature \([7, 8]\).

For a temporal rule \( r^{(k)} \), the support degree of \( r^{(k)} \) at time point \( \tau \) is naturally defined as the number of entity pairs for which the head of \( r^{(k)} \) has instantiations at time point \( \tau \) and the body of \( r^{(k)} \) has instantiation at time point \( \tau - k \). Formally, a pair of entities \( (e, e') \) satisfies the body of \( r \) at time point \( \tau \) with \( \tau \geq 0 \), denoted \( \text{body}(r, e, e', \tau) \), if there exist entities \( e_1, ..., e_{n-1} \) in the KG stream \( S \) such that \( P_1(e, e_1, \tau), P_2(e_1, e_2, \tau), ..., P_n(e_{n-1}, e', \tau) \) are events in \( S[\tau] \). And \( (e, e') \) satisfies the head of \( r \) at time point \( \tau \), denoted \( \text{head}(r, e, e', \tau) \), if \( P(e, e', \tau) \) is an event in \( S[\tau] \). Then the support degree of \( r^{(k)} \) at time point \( \tau \) is defined as

\[
\text{supp}(r^{(k)}, \tau) = \begin{cases} 
0, & \text{if } \tau < k \\
\#(e, e') : \text{head}(r, e, e', \tau) & \land \text{body}(r, e, e', \tau - k), & \text{otherwise}
\end{cases}
\]

(4.1)

Note that, since in the static case there is only one time point 0, the standard notion of support is a special case of the above definition where \( \tau = k = 0 \).

To normalize support degree, the notions of standard confidence and head coverage have been introduced, which correspond to the standard accuracy and recall, respectively. The standard confidence (SC) of a temporal rule \( r^{(k)} \) at time point \( \tau \)
is the ratio between support degree at $\tau$ and the number of entity pairs satisfying
the body at time point $\tau - k$:

$$SC(r^{(k)}, \tau) = \frac{\supp(r^{(k)}, \tau)}{\#(e, e') : body(r, e, e', \tau - k)} \quad (4.2)$$

Similarly, we define head coverage (HC) of a temporal rule $r^{(k)}$ at time point $\tau$ to
be the ratio between support degree and the number of entity pairs satisfying the
head at time point $\tau$:

$$HC(r^{(k)}, \tau) = \frac{\supp(r^{(k)}, \tau)}{\#(e, e') : head(r, e, e', \tau)} \quad (4.3)$$

In the following example, we illustrate the process of temporal rule generation and
the quality measures defined above.

**Example 1.** Consider the first three time points in a KG stream $S$ as follows:

$$S[0] = \{P_1(e_3, e_2, 0), P_1(e_1, e_0, 0), P_1(e_3, e_0, 0),$$
$$P_2(e_3, e_1, 0)\}$$

$$S[1] = \{P_1(e_2, e_2, 1), P_1(e_2, e_1, 1), P_2(e_1, e_3, 1),$$
$$P_2(e_2, e_1, 1), P_2(e_3, e_1, 1), P_2(e_3, e_3, 1),$$
$$P(e_1, e_1, 1), P(e_1, e_3, 1)\}$$

$$S[2] = \{P_1(e_1, e_3, 2), P_2(e_2, e_2, 2), P_2(e_3, e_1, 2),$$
$$P_2(e_2, e_1, 2), P_2(e_3, e_3, 2), P(e_1, e_3, 2), P(e_1, e_1, 2)\}$$

Let $r : P_1(x, z) \land P_2(z, y) \rightarrow P(x, y)$ be a CP rule. Consider $k = 0, 1, 2$ and we
have the following candidate temporal rules based on $r$:

$$r^{(0)} : P_1(x, z, t) \land P_2(z, y, t) \rightarrow P(x, y, t)$$
$$r^{(1)} : P_1(x, z, t) \land P_2(z, y, t) \rightarrow P(x, y, t + 1)$$
$$r^{(2)} : P_1(x, z, t) \land P_2(z, y, t) \rightarrow P(x, y, t + 2)$$
At time point $\tau = 2$, the quality of the these rules can be assessed as follows:

\[
\begin{align*}
\text{supp}(r^{(0)}, 2) &= 2, & SC(r^{(0)}, 2) &= 1, & HC(r^{(0)}, 2) &= 1 \\
\text{supp}(r^{(1)}, 2) &= 0, & SC(r^{(1)}, 2) &= 0, & HC(r^{(1)}, 2) &= 0 \\
\text{supp}(r^{(2)}, 2) &= 1, & SC(r^{(2)}, 2) &= 0.5, & HC(r^{(2)}, 2) &= 0.5
\end{align*}
\]

The SC and HC of temporal rules generalises those standard notions with a temporal flavour, yet they haven’t taken into consideration of the streaming nature of KG stream. In particular, the SC and HC of a temporal rule at one time point is independent from that at other time points. On the other hand, the quality of temporal rules at a time point in a KG stream should aggregate the corresponding (SC and HC) values at previous time points. Hence, the dynamic standard confidence (DSC) of a temporal rule $\gamma$ at time point $\tau$ is defined as follows:

\[
DSC(\gamma, \tau) = \begin{cases} 
SC(\gamma, \tau), & \text{if } \tau = 0 \\
(1 - \alpha) \times DSC(\gamma, \tau - 1) + \alpha \times SC(\gamma, \tau), & \text{otherwise}
\end{cases}
\]  

(4.4)

where $0 < \alpha < 1$ is the learning rate to adjust the weights of previously aggregated DSC and that of the SC of current time point. The dynamic head coverage (DHC) is defined in a similar way.

We use the DSC and DHC scores to select quality temporal rules at each time point. Note that while the set of structure rules are always the same, and so are the candidate temporal rules, the set of selected temporal rules at each point are often different, due to varying DSC and DHC scores of the temporal rules over time.

### 4.3 Combine Stream Learning and Reasoning

In this section, we present our algorithm that combines the learning and reasoning of temporal rules in a dynamic manner over KG streams. Our algorithm takes as
input a KG stream $S$ (i.e., a stream of quadruples), for which all the facts in an initial segment up to time point $n$, $S^*[0,n]$ is stored, and two integers $l,m \geq 0$ as minimum and maximum prediction distances. And it produces as output a stream of temporal rule sets and a stream of derived events. In particular, we use the method from the above section to obtain a set of candidate temporal rules: $r^{(l)}$, $r^{(l+1)}$, ..., $r^{(m)}$ for each structure rule $r$. Then, at each time point, we select quality temporal rules using their DSC and DHC scores and apply the selected rules to derive events about current and future time points.

For both rule quality measure computation and rule application over KG stream $S$, the notion of shifting windows is required. We assume at each time point $\tau$, only a segment of $S$ of size $w$, $S[\tau - w + 1, \tau]$, is used for computation. Here, $w \geq 1$ is an integer called the window size, which may vary over time points. The shifting windows are needed not only for memory space concerns but also due to efficiency requirement of stream processing. Note that the structure rules can be learned offline, whereas the temporal rule filtering and rule application need to be performed online.

At time point $\tau$, to assess the quality of candidate temporal rule $r^{(k)}$, it is clear that if $\tau < k$ then $DSC(r^{(k)}, \tau) = DHC(r^{(k)}, \tau) = 0$; otherwise, we assume $DSC(r^{(k)}, \tau - 1)$ has been obtained from the previous time point. By (4.1) – (4.4), we would need to access the events at time points $\tau$ and $\tau - k$. Yet we can only access the events in the shifting window, that is only those in $S[\tau - w + 1, \tau]$. In this case, if $k \leq w - 1$ then we have required events and the quality measures can be computed as in (4.1) – (4.4); otherwise, we set $DSC(r^{(k)}, \tau) = \beta \times DSC(r^{(k)}, \tau - 1)$ and $HC(r^{(k)}, \tau) = \beta \times HC(r^{(k)}, \tau - 1)$, where $0 < \beta < 1$ is used to adjust the weights of previously aggregated DSC due to the rule’s quality not assessable at the current time point.

At time point $\tau$, to derive new events at time point $\tau + k$ with $l \leq k \leq m$ (recall that $m$ is the maximum prediction distance), we apply selected temporal rule of the form $r^{(k)}$ to the events in $S[\tau]$. To obtain the confidence degree (CD) of a derived event, we adapt the $score^*()$ function from [8] by aggregating the DSC
Chapter 4. StreamLearner

of all the temporal rules that can derive the event in a Noisy-OR manner. The intuition is that events derived by more rules should have a higher confidence degree. Formally, for an event $\xi = P(e, e', \tau)$ and the set of temporal rules $\Gamma$ that can derive $\xi$ from the KG stream, the CD of $\xi$ is defined as follows:

$$CD(\xi) = 1 - \prod_{\gamma \in \Gamma} (1 - DSC(\gamma, \tau)).$$

Figure 4.2: Stream learning and reasoning.

In Figure 4.2, we illustrate one snapshot of the stream where the current point is $\tau$, $w = 4$, and $m = 2$. In this example, we consider one structure rule, $r$. In this time point, we learn the DSC from current and previous time points while we use this time point as the body of rules which get fired and predict the new events for current time point and the future ones.

Algorithm 2 shows the data flow and major components of our system for temporal rule learning and reasoning over KG streams.

In line 2, we obtain a set of structure rules $R$ using static rule learner StaticLearner, such as RLvLR, over the structure training data $S^*[0, n]$. This is performed offline. Then, in line 3, a set $\Gamma$ of candidate temporal rules of the form $r^{(k)}$ is obtained with maximum $k$ being $m$.

The online stream learning and reasoning starts from line 4. In line 5, current events at time point $\tau$ are read in, and in line 7, past events outside of the shifting window are forgotten. Thus, $W$ consists of all the events in the current shifting window.
Algorithm 2 KG Stream Learning and Reasoning

**Input:** A KG stream $S$ and two integers $m, n \geq 0$

**Output:** A set of temporal rules $\Gamma_\tau$ and a set of events $\Xi_\tau$ at each time point $\tau \geq 0$

1: $W := \emptyset, \tau := 0, \Gamma_\tau := \emptyset, \Xi_\tau := \emptyset$

2: $R := \text{StaticLearner}(S^*[0,n])$

3: $\Gamma := \text{CandidateConstruct}(R, m)$

4: **loop**

5: $W := W \cup S[\tau]$

6: **if** $\tau \geq w$ **then**

7: $W := W \setminus S[\tau - w]$

8: **end if**

9: $\Gamma_\tau := \text{Filter}(\Gamma, W)$

10: stream out $\Gamma_\tau$

11: **for** each $l \leq k \leq m$ and each $r^{(k)} \in \Gamma_\tau$ **do**

12: $\Xi_{\tau+k} := \Xi_{\tau+k} \cup \text{Apply}(r^{(k)}, S[\tau])$

13: **end for**

14: stream out $\Xi_\tau$

15: $\tau := \tau + 1$

16: **end loop**

In line 9, candidate temporal rules are filtered based on their DSC and DHC scores at time point $\tau$. As discussed before, this involves aggregating previous DSC and DHC scores, and is restricted by $W$ the events available in the shifting windows. After filtering, the set of selected rules $\Gamma_\tau$ can be streamed out.

The selected temporal rules can then be used for reasoning. In line 12, each rule $r^{(k)}$ is applied to the current events $S[\tau]$ to derive events in future time point $\tau + k$.

Note that events at a time point $\tau$ is derived incrementally from a sequence of past time points. Once all the learnt temporal rules at time point $\tau$ have been applied, the derived events at $\tau$, $\Xi_\tau$, will not change and can be streamed out.

### 4.4 Experiments

We have implemented a system, StreamLearner, based on the above algorithm and conducted several experiments to evaluate it. In our implementation, RLvLR
(chapter 3) is deployed for learning structure rules. The executable codes, benchmark datasets and experimental results are publicly available at https://www.ict.griffith.edu.au/aist/RLvLR/.

Our experiments are designed to demonstrate that temporal rules are an effective model to capture temporal knowledge and thus can provide more accurate link prediction than some existing statistical models, such as TransE [16] and HOLE [15], that do not account for temporal knowledge or evolving data. Our goal is not to compete with temporal statistical models like Know-Evolve [9] and [48] and due to the infeasibility of running their codes, we do not include them in our experiments. To analyse the benefit of having temporal rules over static rules in stream reasoning, we also used a “static” version of StreamLearner (StreamLearner-S), which only uses the static structure rules (not the temporal rules). Specifically, our experimental results aimed to validate the following observations:

1. StreamLearner significantly outperforms the baseline methods StreamLearner-S, TransE and HOLE in terms of accuracy in link prediction.

2. Stream learning of and link prediction through temporal rules can be performed efficiently. When the sizes of structure training data increased, the prediction accuracy also increases without significant sacrifice on the efficiency.

3. The accuracy of link prediction generally increases when the sizes of shifting window increases or when the minimum prediction distances reduces. StreamLearner still outperforms StreamLearner-S in the cases of small window sizes and long minimum prediction distances.

In our experiments, we used the Integrated Crisis Early Warning System (ICEWS) dataset [9], which is based on historical events including interactions between socio-political agents (i.e., cooperative or antagonistic actions between individuals, associations, organizations, sectors and nation states). The events were automatically recognized and extracted from the news. It contains events from the year of 2014 with a 24hrs interval between each two adjunct time points (that is, 365 time
points in total). ICEWS is an ideal benchmark for our experiments because the behaviour of agents in this benchmark has complex interaction patterns. We are unaware of any other benchmarks of a similar type. We also adopted the ICEWS-500 as in [9], which contains a small subset of ICEWS with 500 entities. We included this small subset to examine the impact of data sizes on different systems. Some statistics about the two datasets are shown in Table 4.1, including the numbers of entities, numbers of events, numbers of predicates, and numbers of time points.

### Table 4.1: Dataset statistics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Entity</th>
<th>#Event</th>
<th>#Pred.</th>
<th>#TPoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICEWS</td>
<td>12498</td>
<td>668080</td>
<td>260</td>
<td>365</td>
</tr>
<tr>
<td>ICEWS-500</td>
<td>500</td>
<td>445665</td>
<td>260</td>
<td>365</td>
</tr>
</tbody>
</table>

We have conducted two sets of experiments. In the first set of experiments, we used the facts from the first 50 days (i.e., $S^*[0, 50]$) as the training data for learning structure rules. We note that the size of the data set is much smaller than the initial data sets in other approaches for learning from streams. Since static rule learner RLvLR learns static rules about specified predicates in rule heads, called target predicates, we randomly selected 20 target predicates.

We assessed the accuracy of link prediction fortnightly after the 50th day. That is, 23 time points after the rules structure training. For each of these time points $k$, we separated the datasets into 70% training and 30% testing as in most literature. The link prediction task is to identify for each target predicate $P$ and each entity $e$, an entity $e'$ such that $P(e, e', k)$ is an event occurred in the KG stream; or alternatively, to identify for each target predicate $P$ and each entity $e$, an entity $e'$ such that $P(e', e, k)$ is an event in the stream. The prediction accuracy was measured by filtered Mean Reciprocal Rank (MRR) and Hits@10 as in the literature [16].

In this experiment, the size of shifting window was fixed to 10 days and the maximal prediction distance was also 10 days. For each time point $k$, StreamLearner applied the learnt temporal rules to the training data in $S[k]$, as well as
Chapter 4. *StreamLearner*

$S[k - 10, k - 1]$, to derive missing events at $k$, whereas StreamLearner-S applied all the structure rules to the training data in $S[k]$. TransE and HOLE obtained their respective statistical models from the training data in $S[k]$.

Table 4.2 shows the performance of StreamLearner, StreamLearner-S, TransE and HOLE. We report the numbers of temporal rules (#R), the numbers of structure rules (#SR), MRR and Hits@10 (H@10) scores, averaged over all selected time points and target predicates.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>StreamLearner</th>
<th>StreamLearner-S</th>
<th>TransE</th>
<th>HOLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#TR  MRR</td>
<td>#SR  MRR</td>
<td>MRR</td>
<td>MRR</td>
</tr>
<tr>
<td>ICWEC</td>
<td>1748 0.35 0.48</td>
<td>568 0.24 0.30</td>
<td>0.05  0.12</td>
<td>0.07  0.10</td>
</tr>
<tr>
<td>ICWEC-500</td>
<td>1728 0.37 0.54</td>
<td>535 0.22 0.31</td>
<td>0.05  0.14</td>
<td>0.05  0.09</td>
</tr>
</tbody>
</table>

The experimental results show that StreamLearner significantly outperformed the baseline systems. In particular, StreamLearner was around 4 times more accurate than TransE and HOLE on Hits@10, and about 5 times more accurate on MRR. StreamLearner also showed superior performance to its static version, which clearly demonstrates the benefit of temporal rules over static rules on stream reasoning. Indeed, the average number of temporal rules is only around 3 times more than that of structure rules. Considering the number of candidate temporal rules (10 times of structure rules), it shows that the temporal rules can model much refined and more precise association among events.

Figure 4.3 shows the detailed performance of StreamLearner and the baseline systems, from which we can see that of StreamLearner showed consistently superior performance over the time.

A major benefit of temporal rules compared to statistical models is that their meanings are human understandable, and hence the learned temporal rules themselves contain valuable temporal knowledge about the domain. Although entities and predicates in the ICWEC dataset are obfuscated and thus it is difficult to assess the meaning of learnt temporal rules, it is not hard to image the temporal knowledge behind some common patterns. For illustration purpose, we present the
following two temporal rules whose predicate names are replaced with meaningful terms from the ICEWS dictionary.

\[ 0.05 : \text{rejectMaterialCooperation}(y, x, t) \rightarrow \text{threatenWithSanctions}(x, y, t). \]
\[ 0.15 : \text{rejectMaterialCooperation}(y, x, t) \rightarrow \text{threatenWithSanctions}(x, y, t + 1). \]

The number before each rule is the corresponding DSC. The two rules indicate that if country \( y \) rejects the material cooperation with country \( x \), then it is unlikely that \( y \) threatens \( x \) with sanctions on the same day. But this is more likely to happen on the next day.

The quality of learnt temporal rules and their performance in link prediction are affected by several factors related to the KG streams, including the sizes of structure training data, the window sizes, and the (minimum and maximum) prediction distance. For the second set of experiments, we evaluate how these factors affect the quality and performance of temporal rules.

The size of structure training data affects the quantity and quality of the structure rules, which in turn affects the learning of temporal rules. In particular, it impacts the efficiency of StreamLearner. Table 4.3 shows the performance of StreamLearner with different sizes \( n \) of structure training data \( S^*[0, n] \), and we compare its performance with that of StreamLearner-S. We report the numbers of structure rules (#SR), stream learning and reasoning times (Time, in seconds), MRR and Hits@10 (H@10) scores, averaged over all selected time points and target predicates.

Overall, allowing larger structure learning data provides more structure rules and leads to better prediction accuracy on both StreamLearner and StreamLearner-S. Yet, there is trade-off between time efficiency and prediction accuracy, and as the size of structure learning data increases, the processing time of StreamLearner increases more significantly than that of StreamLearner-S, due to the number of
Table 4.3: Performance on varying sizes of structure learning data

<table>
<thead>
<tr>
<th>Size</th>
<th>#SR</th>
<th>StreamLearner</th>
<th>StreamLearner-S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time MRR H@10</td>
<td>Time MRR H@10</td>
</tr>
<tr>
<td>10</td>
<td>27</td>
<td>0.8 0.19 0.25</td>
<td>0.1 0.02 0.02</td>
</tr>
<tr>
<td>50</td>
<td>123</td>
<td>3.3 0.22 0.30</td>
<td>0.2 0.06 0.11</td>
</tr>
<tr>
<td>100</td>
<td>262</td>
<td>7.0 0.27 0.37</td>
<td>0.4 0.10 0.19</td>
</tr>
<tr>
<td>150</td>
<td>355</td>
<td>9.6 0.26 0.38</td>
<td>0.6 0.12 0.21</td>
</tr>
<tr>
<td>250</td>
<td>551</td>
<td>14.1 0.30 0.44</td>
<td>0.8 0.14 0.24</td>
</tr>
</tbody>
</table>

Candidate temporal rules. Yet the processing time of StreamLearner remains acceptable (14.1 seconds) even when a significantly large portion of the KG stream (250 out of 365) were used as structure training data. While StreamLearner again significantly outperformed StreamLearner-S in all cases, an interesting observation is when the available data for rule structure learning is smaller, the benefit of using temporal rules over static rules is more obvious. This indicates when the available structural knowledge (i.e., association among facts, represented in structure rules and temporal rules) is relatively limited, the temporal knowledge (i.e., association among events, represented in temporal rules) plays a more significant role in prediction.

The window sizes and the maximum prediction distances both determine the amount of historical data that can be utilised for learning and prediction, and thus affect the performance of our system. For simplicity, in the following experiments, we set the window sizes to be fixed over time and equivalent to the maximum prediction distances. Figure 4.4 shows the performance of StreamLearner over window sizes ranging from 1 to 15. Again, we used StreamLearner-S as a baseline, whose performance is not impacted by the window sizes, and MRR and Hits@10 scores are averaged over all selected time points and target predicates.

Generally speaking, the performance of StreamLearner improves as the window size increases. It also reveals some local optimal points, namely window sizes 3 and 10. Although slight improvement in Hits@10 can be observed for window sizes larger than 10, MRR seems to be flattening. This suggests that in real-life, events occurring further way in the stream tend to be more loosely associated with the
current events, and in practice it could be effective to use small window sizes (like 3 or 10).

The minimum prediction distance also impacts the learning and prediction, and probably more on prediction. In the previous experiments, we set the minimum prediction distance to be 0, which allowed events to be derived from the current data. Clearly, as the minimum prediction distance increases, the system is challenged to finalise the derived events in distance. For instance, if the minimum prediction distance is set to 3 then the learnt temporal rules are of the form $r^{(3)}, r^{(4)}, \ldots$. That means, all derived events at time point $\tau$ ($\Xi_\tau$) are obtained at time point $\tau - 3$. This effectively requires the prediction to happen at least 3 days ahead.

In the following experiments, we evaluated the impact of minimum prediction distance on the performance of StreamLearner, using StreamLearner-S as a baseline. Note that if the maximum prediction distance stays the same, the increase of minimum prediction distance will reduce the number of candidate temporal rules. To separate this factor from the challenged post by distant prediction, we assume the difference between the maximum and minimum prediction distance remains 10. Figure 4.5 shows the performance of StreamLearner over minimum prediction distances ranging from 0 to 10. Again, the values are averaged.

The performance of StreamLearner drops as the prediction distances increase. An interesting observation is that the accuracy decrease was not as dramatic as one would expect. In particular, predicting one day or two days ahead had comparable accuracy as “predicting” about today. Also, 7 turns out to be a local optimal point, which may suggest weekly pattern in the event association.
4.5 Related Works

The topic of this chapter is in the intersection of stream reasoning and stream learning. There have been some approaches to address these two issues but separately. Knowledge representation and reasoning in the presence of data streams has been investigated in [11, 46, 47, 69]. As we have reviewed them in section 2.2.2, they did not investigate how to extract such rules from data streams. There are numerous works on link prediction by employing the method of embedding such as [9]. Our method for rule learning from KGs is different from theirs, while our method can also be applied in link prediction. This is because we focus on learning structural information in the form of first order rules. There are also some approaches to learning logic-based models form streams of relational data such as [70, 71]. A method is proposed in [70] for extracting event definitions automatically. Their system incrementally refine the logical definition of events based on temporal events information dataset. As the method is based on refinement operators in inductive logic programming, the resulting system is not scalable for handling KGs.

4.6 Conclusion

In this chapter, we have proposed a method for learning temporal rules from data streams in the format of Knowledge Graphs (KGs). Such rules can be used for reasoning about event over different time points. Especially, temporal rules are useful for link prediction and event prediction in the setting of KG data streams. The proposed method is essentially a framework for rule learning from KG data streams by extending existing rule learners for KGs. Based on RLvLR (chapter 3), our efficient rule learner for KGs, we have implemented a system StreamLearner, which can learn temporal rules from KG data streams, completing dynamic KGs and predicting events for them. Our experiments show that StreamLearner significantly outperforms the base systems. It is able to handle large KGs and a
promising approach to creating, maintaining and using data streams in the format of KGs.
Figure 4.3: Performance comparison on link prediction over KG streams.
Chapter 4. \textit{StreamLearner}

Figure 4.4: Performance over varying window sizes.

Figure 4.5: Performance over varying minimum prediction distances.
Chapter 5

Learning Rules via Transfer Learning from KGs

In the previous chapters, we tackle learning logical rules from KGs under different scenarios. In this chapter, we attempt to address the sparsity of data in KGs by utilizing transfer learning concept.

Learning rules from knowledge graphs is an important yet challenging task, especially when the given data is sparse. Transfer learning is an actively researched area to address the sparse data issue, which learns a predictive model for the target domain from a similar source domain. We propose a transfer learning approach, named Transfer Rule Learner (TRL). TRL deploys embedding representations of KGs to select most relevant knowledge graphs and obtain a mapping between the predicates of these KGs. TRL uses the obtained mapping to learn the new rules.

We conducted the experimental evaluation on standard knowledge graphs, and the results show our system can obtain quality rules even with extremely sparse data and its predictive accuracy outperformed the state-of-the-art rule learners (AMIE+ and RLvLR) and link prediction systems (HOLE and TransE).

This chapter resulted in the following work:

Pouya Ghiasnezhad Omran, Zhe Wang, Kewen Wang: Knowledge Graph Rule Mining via Transfer Learning. In PAKDD 2019 [accepted for publication]
5.1 Introduction

Despite of the sizes of large-scale KGs (e.g. Freebase [4]), the issue of data sparsity exists, that is, when only a small number of relevant facts are available. For example, while large KGs contain many entities, most entities are associated with a sparse set of facts [72], and for a large portion of predicates, only a few facts are available for each of them. For instance, only few facts are available for the predicates, organismsOfThisType and exhibitionSponsor in Freebase (FB15k). Also, when companies develop their business related KGs, these KGs often need to be developed from scratch and from small sizes, where both entities and available facts are few.

The importance of coupling KGs with logical rules has been highlighted in the previous chapters. However, the existing approaches to KG rule learning (e.g. ScaleKB [7], AMIE+ [8] and RLvLR (Chapter 3)) all suffer from the so-called data starvation problem [12]. That is, they require very large dataset for rule learning and struggle to learn over sparse data.

Transfer learning is a promising paradigm for overcoming the data starvation problem. The goal of transfer learning is to reuse knowledge learned from one task (source task) in a different and unlearned task (target task). This paradigm of learning is mostly pursued in feature vector machine learning, but some attempts have been made to learn more complex models. These methods usually assume that the source and target domains are highly similar in terms of models of interest. Once such strict conditions are violated, unfavourable effects can be caused. This is usually referred to as negative transfer [12, 49].

In this chapter, we tackle the problem of rule learning with sparse data by employing transfer learning. Given a KG, a learning task (or just task) is that of learning rules whose head is about a specific predicate. Given a set of source tasks and a target task, we assume that each source task is learned, that is, a set of rules have been learned for the source task, and we aim to learn a set of rules for the target task based on the source tasks. To facilitate knowledge transfer and reduce the
effect of negative transfer, we make use of embedding techniques in representation learning.

The basic idea of embedding techniques is to encode relational information as low-dimensional representations (embeddings) of entities and predicates. Such representation learning techniques have been applied in rule learning in KGs [26, 44]. Instead of directly learning rules from data and embeddings, we utilize the pre-trained embeddings to measure structural similarity between source and target tasks, in order to transfer logical rules from source tasks to the target task.

The novelty of our approach lies in that the paradigm of transfer learning and embedding techniques are combined for rule mining which is not trivial. Transfer learning can help address the data starvation problem while embedding techniques can provide useful similarity measures that reduce negative transfer. As a result, this combination makes it possible to develop effective systems for rule mining over sparse data in KGs. In fact, our experiments demonstrate that on sparse data, our system outperformed state-of-the-art rule learner AMIE+ [8] and RLvLR [73] in terms of the number of quality rules and the accuracy in link prediction. We also compared our system with embedding-based link prediction systems, our system outperformed TransE [16] and HOLE [15] in link prediction on sparse predicates regarding accuracy and runtime.

The chapter is organised as follows. We provide an overview of our proposed method in Section 5.2. We define measures for similarity using embeddings in Section 5.3. The experimental evaluation is reported in Section 5.4. Finally, we discuss the relation of our approach to some other approaches in Section 5.5 and conclude our work in Section 5.6.

5.2 An Overview of Our Approach

In a traditional transfer learning setting, a pair of source and target domains are considered, and the learner uses knowledge from the source domain to learn
knowledge in the target domain. Certain (structural) similarity between the two
domains is assumed, which is critical for the quality of learning.

To learn rules over KGs, we relax such an assumption in two ways. First, we do
not assume the source domain for transferring is known; instead, we generate a
pool of potential sources from existing KGs and measure their similarity with the
target and the source with highest similarity to the target is selected for transfer.
Second, since existing KGs are large and comprehensive knowledge bases, to learn
rules about certain predicates in a KG, we can use available rules about other
predicates within the same KG as sources for transferring.

To achieve these, we first define a learning task (or simply a task) $T$ to be a triple
$T = (P, R, K)$ where $K$ is a KG, $P$ is a predicate in $K$ and $R$ is a set of rules about
$P$ (i.e., having $P$ in its head) over $K$. Intuitively, $T = (P, R, K)$ represents the task
of learning rules about predicate $P$ over KG $K$, and $R$ consists of the rules learnt so
far. For transfer learning, the input consists of a target task $T_t = (P_t, R_t, K_t)$ and
a set of (potential source) tasks $T_i = (P_i, R_i, K_i)$ with $1 \leq i \leq n$. For convenience,
we assume $R_t = \emptyset$, each $R_i$ consists of all the rules about $P_i$ in $K_i$, $P_i \neq P_j$, and
$P_t \neq P_i$ for all $1 \leq i < j \leq n$. We call $P_i$ the goal predicate for task $T_i$. We do
not assume any intersection or disjointness between $K_t$ and $K_i$’s. The output of
learning is a set of rules $R_t$ about $P_t$ over $K_t$.

Example 2. Consider a KG $K$ containing predicates above movies, \{producedIn, actedIn, livesIn\}, then each predicate corresponds to a task and the task about
actedIn has the form $(actedIn, R, K)$ where $R$ consists of all the rules about actedIn
over $K$. Suppose we want to learn rules about football clubs with the predicates
\{isLocatedIn, playsFor, wasBornIn\} in KG $K'$ (not necessarily different from $K$). A
target task about playsFor is of the form $(playsFor, \emptyset, K')$.

In what follows, we present our embedding-based approach to rule learning in a
nutshell. Following the approach in analogical reasoning [74], our method con-
sists of four phases: (1) Retrieval: From the set of source tasks (from different
domains), we select those tasks that are most similar to the target domain. Ob-
vously, a major challenge here is to define a measure for similarity of learning
tasks. We resolve this issue by employing a Frobenius norm based on predicate embeddings. Notice that the similarity of learning tasks is essentially a kind of local similarity. Thus we do not require the whole source domain is similar to the whole target domain. (2) Mapping: After determining a source task, we map two tasks predicates regarding their structural similarity. Based on the new similarity degree, for each predicate in the target task, we are able to identify the set of predicates in source tasks that are closest to the target predicate. In this phase, three transfers are introduced, linear transformation, translation and distance from goal predicate. (3) Transferring: crafting target task rules from the source rules and the obtained mapping. (4) Validating: we evaluate the transferred rules against the given sparse data for target task. To do so we assess the number of instances of each rule; if the evaluation suggests promising result, we consider the transferred rules as valid rules.

In Algorithm 3, we illustrate the data flow and major components of our algorithm.

**Algorithm 3** Transfer learning for rule learning in KGs

**Input:** a target task $T_t = (P_t, \emptyset, K_t)$ and a set of tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ where $T_i = (P_i, \mathcal{R}_i, K_i)$ for $1 \leq i \leq n$

**Output:** a set of rules $\mathcal{R}_t$ about $P_t$ over $K_t$

1: $(E_t, E_1, \ldots, E_n) := \text{Embeddings}(K_t, K_1, \ldots, K_n)$
2: $\mathcal{T}' := \text{Retrieve}(P_t, P_1, \ldots, P_n, E_t, E_1, \ldots, E_n)$
3: $\mathcal{R}_t := \emptyset$
4: for each mapping method $\text{Map}_k$ do
5: for each $T_i \in \mathcal{T}'$ do
6: $\mathcal{M} := \text{Map}_k(E_t, E_i)$
7: $\mathcal{R} := \text{Transfer}(\mathcal{R}_t, \mathcal{M})$
8: $\mathcal{R}_t := \mathcal{R}_t \cup \text{Validate}(\mathcal{R}_t, K_t)$
9: end for
10: end for
return $\mathcal{R}_t$

In line 1, the \text{Embeddings} method computes the embeddings for all tasks (i.e. source tasks and target task). For each task which the KG (say $K_i$) is too large and RESCAL cannot handle it, we employ a sampling method which produces a fragment of the KG consisting of a bounded number of entities that are related to the goal predicate (i.e., $P_i$) of $T_i$. In this case, $K_i$ is replaced by its small sampling [73]. After sampling, we compute predicate embedding as well as subject and
object argument embeddings for all predicates which exist in the KG of task after sampling as it is done in [73]. Thus, a specific predicate in a specific KG has multiple embedding representations regarding the different tasks (different goal predicate).

After that, in line 2, the Retrieve method selects a set of tasks $T'$ from the pool of tasks $T$ that are considered most similar to the target task $T_t$. Tasks in $T'$ are called source tasks. Since our goal is to learn rules about $P_t$, we consider a task $T_i$ similar to $T_t$ if the goal predicate $P_i$ (of $T_i$) is similar to $P_t$, and such similarity is measured by their embeddings. Since each predicate embedding contains information about the entities and predicates that are related to the predicate and how they are related, similarity between predicate embeddings suggest similarity between predicates in their respective KG contexts.

Then, in lines 4 and 5, we consider different mapping approaches and different source tasks respectively. In line 6, for each source task $T_i$, the Map$k$ method establishes a mapping $M$ between the predicates in respectively the source KG $K_i$ and the target KG $K_t$. The mapping is built under the assumption that the goal predicate $P_i$ of $T_i$ is mapped to the goal predicate $P_t$ of $T_t$. Again, the mapping is established through the corresponding embeddings in $E_i$ and $E_t$. We introduce three methods for building such mappings ($1 \leq k \leq 2$) which we will explain in detail in Section 5.3. After we obtain the mapping, $M$, between predicates in the two tasks, the Transfer method transfers the source rules in $R_i$ to generate candidate rules for $R_t$, by substituting the source predicates of a rule with the mapped target predicates, in line 7.

Finally, in line 8, the Validate method validates the transferred rules and returns the valid rules to be added to $R_t$. Note that validation is challenging over sparse data, where both SC and HC have limited significance. Instead of merely validating each candidate rule one after another, which is ineffective due to the data sparsity, we measure for each source task $T_i$ the percentage of transferred rules that have at least one support in the target KG $K_t$. We call this measure Transfer Confidence (TC), and it essentially validates the selected source task and the established
mapping. As a result, when the TC is above a given threshold, all the transferred rules (obtained from \( T_i \)) are considered valid. We define TC as follows:

\[
TC(\mathcal{R}, \mathcal{K}_t) = \frac{\#r \in \mathcal{R} : \text{supp}(r) \geq 1}{|\mathcal{R}|}
\]

One might think having minimum support of 1 is rather weak, but this measure turns out to be effective for transfer learning (considering the distance of the two tasks with distinct predicates and possibly different KGs). Our experimental results confirmed the effectiveness of this measure.

5.3 Similarity Measure via Embeddings

In this section, we present our approaches for measuring the similarity between two predicates (in possibly two different KGs), which is a key step of transfer learning for rule learning in KGs. These measurements are also central to both Retrieve and Map methods in Algorithm 3.

5.3.1 Similarity Between Goal Predicates

In the method Retrieve, we need to select source tasks \( T_i = (P_i, \mathcal{R}_i, \mathcal{K}_i) \) that are considered most similar to the target task \( T_t = (P_t, \mathcal{R}_t, \mathcal{K}_t) \), in order to transfer rules about \( P_i \) to obtain candidate rules about \( P_t \). Intuitively, this requires the goal predicate \( P_i \) of \( T_i \) to be structurally similar to the goal predicate \( P_t \) of \( T_t \), that is, how similar these two predicates associate entities in their respective KGs. In this following example, we show that such structural similarity can be captured by their predicate embeddings.

Example 3. Consider Example 2, to measure the similarity between the two tasks \((\text{actedIn}, \mathcal{R}, \mathcal{K})\) and \((\text{playsFor}, \emptyset, \mathcal{K}')\), it boils down to the measurement of the structural similarity between the two predicates \text{actedIn} and \text{playsFor}. Intuitively, structural similarity can be understood as follows: Considering \( \mathcal{K} \) and \( \mathcal{K}' \) as graphs, take the subgraphs \( G \) and \( G' \) of \( \mathcal{K} \) and \( \mathcal{K}' \) consisting of only edges labelled with
actedIn and playesFor respectively, then the structural similarity between the two (non-labelled) graphs $G$ and $G'$ reflects the structural similarity between the two predicates actedIn and playesFor.

Note that the embeddings of actedIn and playesFor, as $k \times k$ square matrices, where $k$ is a parameter set by representation learners like RESCAL, capture exactly how entities (in $\mathcal{K}$ and $\mathcal{K}'$) are associated respectively by these two predicates. In particular, $G$ and $G'$ can be constructed from these embeddings.

For two matrices $\mathbf{M}_1$ and $\mathbf{M}_2$, the closeness between them is defined in the standard way as the Frobenius norm:

$$
close(\mathbf{M}_1, \mathbf{M}_2) = -\|\mathbf{M}_1 - \mathbf{M}_2\|_F.
$$

We also consider the number of predicates in the task after target oriented sampling to do so we consider the following equation:

$$
cl(T_i, T_t) = \|\#P_i - \#P_t\|
$$

Where $\#P_i$ and $\#P_t$ are the numbers of predicates in the tasks $i$ and $t$ after sampling respectively. Then, the similarity degree between source tasks $T_i = (P_i, \mathcal{R}_i, \mathcal{K}_i)$ and target task $T_t = (P_t, \mathcal{R}_t, \mathcal{K}_t)$ can be defined as

$$
sim(T_i, T_t) = \exp(close(P_i, P_t)) - \alpha \times cl(T_i, T_t)
$$

where $P_i$ and $P_t$ are embeddings of $P_i$ and $P_t$ respectively. Besides, $\alpha$ is a small coefficient (e.g. 0.01) which imposes the effect of similarity regarding the number of predicates.

The Retrieve method computes the similarity degrees between each pair of $T_i$ and $T_t$ and select source tasks that with highest similarity degrees to the target task.
5.3.2 Similarity Between Other Predicates

For a pair of source task $T_i = (P_i, R_i, K_i)$ and target task $T_t = (P_t, R_t, K_t)$ where $P_i$ and $P_t$ are deemed to be similar, to transfer rules concerning KG $K_i$ and construct rules concerning KG $K_t$, the Map method builds a mapping between the predicates in respectively $K_i$ and $K_t$. To achieve this, we measure the similarity between each pair of predicates in respectively $K_i$ and $K_t$. Such a notion of similarity should consider not only how the concerned predicates associate entities (as in the previous section), but also how they relate to the goal predicates and the other predicates. Also, the measure needs to assign highest similarity degrees to the goal predicates $P_i$ and $P_t$, that is, it should be a natural extension of the measure presented in the previous section.

There are several different approaches introduced in [75] for entity alignment and we found the following one most relevant to our case: linear transformation and translation.

**Linear Transformation**

In this approach, we use a single matrix, $T$, to represent the transformation from the source goal predicate to the target goal predicate, and then apply this transformation to each source predicate to search for the corresponding target predicate. Formally, for a pair of source and target tasks $T_s, T_t$ with goal predicates $P_s, P_t$, a transformer $T$ from $P_s$ to $P_t$ satisfies the following condition:

$$P_s \cdot T = P_t$$

That is, $T = P_s^{-1} \cdot P_t$ whenever the matrix $P_s$ is invertible, which is the case for all matrices we had in our experiments. If $P_s$ is not invertible, we can simply omit this source task and use other source tasks.

Then, we apply the transformer to each source predicates and find the target predicates that are closest to the result of transformation. For two predicates $P$
and $P'$ from respectively source task $T_s$ and target task $T_t$, their similarity degree is defined as follows:

$$\text{sim}_{LT}(P, P') = \text{close}(P \cdot T, P').$$

### Translation with Argument Embeddings

In this approach, we use the argument embeddings besides predicate embeddings. We also use additive calculus instead of the multiplication which we use in the previous approach. The argument embeddings of predicates are computed as we explained in section 3.3. In this case we have three translations as following:

$$P_s + T = P_t$$
$$P_s^s + T^s = P_t^s$$
$$P_o^o + T^o = P_t^o$$

Then, we apply the translation to each source predicate and find the target predicate that is closest to the result of transformation. For two predicates $P$ and $P'$ from respectively source task $T_s$ and target task $T_t$, their similarity degree is defined as follows:

$$\text{sim}_{TAE}(P, P') = \text{close}(P + T, P') + \text{close}(P_s + T^s, P'_s) + \text{close}(P^o + T^o, P'^o).$$

These two approaches are shown in Figure 5.1.

The $\text{Map}_k$ method computes the similarity degrees, regarding to different $\text{sim}$ methods, between each pair of predicates from source and target tasks, and builds a mapping that maps each source predicate to a target predicate with the highest similarity degree.

Regardless of the method which we compute the similarity between source predicates and target predicates, we use a greedy global maximising method to obtain the mapping. To do so, we assign the mapping for the pair of predicates which have the maximum similarity. Then, we eliminate these two predicates and continue with the next pairs with maximum similarity and so on. In some cases, the
number of source predicates is more than the number of target predicates. In this case, for some source predicates, we do not have any correspondent from the target predicates. In transforming rules, we ignore all rules which have such predicates.

5.4 Experiments

We have implemented a Transfer Rule Learner (TRL) based on the described algorithms and conducted two sets of experiments on inter- and intra-KG transfer rule learning. The datasets and detailed results can be found at https://www.ict.griffith.edu.au/aist/TRL/.

The adopted benchmark datasets for our experiments include modified versions of Freebase and YAGO. The two benchmark datasets are specified in Table 5.1. FB15K [16] is the same as used in [14], and YAGO2s is the same as in [8].

<table>
<thead>
<tr>
<th>Table 5.1: Benchmark specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>KG</td>
</tr>
<tr>
<td>FB15K</td>
</tr>
<tr>
<td>YAGO2s</td>
</tr>
</tbody>
</table>
Since existing transfer rule learners cannot handle the adopted datasets (as we will explain in Section 5.5), we compared TRL with state-of-the-art non-transfer rule learners AMIE+ [8] and RLvLR (Chapter 3), as well as statistical link predictors TransE [16] and HOLE [15]. Our experiments were designed to validate the following statements:

1. For small KGs (i.e., with small numbers of facts), TRL is able to learn, through inter-KG transfer, more quality rules than AMIE+ and RLvLR.

2. For sparse predicates (i.e., associated with limited numbers of facts) in large KGs, TRL can also provide, through intra-KG transfer, better accuracy in link prediction than RLvLR, TransE and HOLE.

All experiments were conducted on a desktop with Intel Core i5-4590 CPU at 3.3GHz (one thread) and with 8GB of RAM, running Ubuntu 14.04.

### 5.4.1 From FB15K to YAGO2s

For the first set of experiments, we evaluated inter-KG transfer rule learning from large source KGs to small target KGs. To this end, we pruned YAGO2s by eliminating entities and predicates with fewer than respectively 10 and 100 occurrences, and the pruned version contains 24 predicates. For each of these 24 predicates \( P_t \), we constructed a target task with \( P_t \) as the goal predicate as follows: we first set apart 30\% of the facts about \( P_t \) as the test data and the remaining facts as evaluation data. From the evaluation data, we extracted three subsets of YAGO2s with decreasing sizes, i.e, with 500, 200 and 100 entities, as the target KGs, through a sampling procedure similar to that in section 3.2.

To obtain a pool of potential source tasks for transfer, we coupled FB15K with rules learnt by RLvLR. For generating source rules, other rule learner like AMIE+ can be also used, and we deployed only RLvLR for implementation convenience.

For each target task, top 100 most similar source tasks were selected from the pool for transfer.
Table 5.2: Transfer rule learning from large KGs to small KGs

<table>
<thead>
<tr>
<th>#E</th>
<th>#P</th>
<th>#F</th>
<th>TRL</th>
<th>AMIE+</th>
<th>RLvLR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>#QR</td>
<td>MRR</td>
<td>H@10</td>
</tr>
<tr>
<td>500</td>
<td>17.2</td>
<td>44K</td>
<td><strong>16.3</strong></td>
<td>0.18</td>
<td>30</td>
</tr>
<tr>
<td>200</td>
<td>12.6</td>
<td>17K</td>
<td><strong>13.6</strong></td>
<td>0.17</td>
<td>28</td>
</tr>
<tr>
<td>100</td>
<td>11.3</td>
<td>6K</td>
<td><strong>10</strong></td>
<td>0.17</td>
<td>29</td>
</tr>
</tbody>
</table>

We compared TRL with AMIE+ and RLvLR, where those two learned rules directly from the target KGs. To evaluate the quality of learnt rules, we first evaluated their SC and HC over the evaluation data. Note that the evaluation data are not sparse, and thus the SC and HC scores indeed reflect the quality of the learnt rules. We recorded the numbers of rules with SC ≥ 0.1 and HC ≥ 0.01.

We also evaluated the quality of rules through link prediction, through two queries $P_t(e, ?)$ and $P_t(?, e')$ for each goal predicate $P_t$ and each fact $P_t(e, e')$ from the test data. We applied learnt rules on the evaluation data to predict the missing entities (instead of the sparse data in the target KGs for all three learners) for better evaluation of the predictive power of the learnt rules. The predictions were measured using Mean Reciprocal Rank (MRR) and Hits@10, by ranking each inferred fact based on the numbers of rules inferring it [14].

Table 5.2 summarizes the results, where #E, #P and #F are respectively the numbers of entities, predicates and facts in the target KGs, #QR is the number of quality rules, and H@10 is the Hits@10 score in percentage. All the numbers are averaged over the 24 target tasks.

While AMIE+ struggled on such sparse data (learning averagely fewer than a single rule), TRL managed to learn more than 10 rules on average. On relatively larger KGs with 500 entities, the predictive power of rules produced by RLvLR was comparable to those produced by TRL. Yet as the sizes of target KGs decrease, the performance of both AMIE+ and RLvLR dropped significantly, whereas that of TRL is relatively stable due to the nature of transfer learning.

Among the rules learnt by TRL, we discovered some interesting rule patterns (i.e., transferable knowledge). For instance, a large number of rules are like the following
rules (with distinct predicates):

\[ \text{wasBornIn}(x, z) \land \text{wasBornIn}(t, z) \land \text{livesIn}(t, y) \rightarrow \text{livesIn}(x, y). \]

which states if two persons \( x \) and \( t \) were born in the same city \( z \) then it is likely that they both live in the same city \( y \). There are also quite a few rules stating symmetry property of predicates such as

\[ \text{influences}(y, x) \rightarrow \text{influences}(x, y). \]

and those stating association between predicates such as

\[ \text{isAffiliatedTo}(x, y) \rightarrow \text{playsFor}(x, y). \]

### 5.4.2 From FB15K to FB15K

For the second set of experiments, we evaluated TRL on intra-KG transfer learning, i.e., to transfer rules from rich data to sparse data within the same KG. FB15K includes a large number of predicates that are each associated with only a small number of facts, which we call sparse predicates. We consider the top 5\%, 10\%, 15\%, and 20\% most sparse predicates in FB15K from all of its 1345 predicates, and use them as goal predicates \( P_t \). FB15K comes with its own separate training and test data [16], and we used its training data for both the target KG and the source KG. We eliminated from the pool of source tasks those with same goal predicates as \( P_t \) to avoid trivial transfer.

Since RLvLR outperforms AMIE+ on large KGs (section 3.5) we compared TRL with RLvLR and focused on link prediction. We added two statistical link predictors (not rule learners) TransE and HOLE in the comparison, and measure the prediction only against goal predicates \( P_t \) (by removing irrelevant facts in test data). For fine-grained comparisons of prediction accuracy, we also measured the Hits@1 and Hits@3 scores, which are the proportions of correctly predicted entities that are ranked respectively, top one and top three.
Table 5.3: Transfer rule learning from dense predicates to sparse predicates

| %P | #P | TRL | | | | RLvLR | | | |
|---|---|---|---|---|---|---|---|---|
| | | Time | MRR | H@1 | H@3 | H@10 | Time | MRR | H@1 | H@3 | H@10 |
| 20% | 272 | 3.5 | 0.18 | 13 | 23 | 24 | 6.8 | 0.14 | 9 | 16 | 21 |
| 15% | 204 | 2.2 | 0.17 | 13 | 20 | 22 | 3.9 | 0.11 | 7 | 12 | 17 |
| 10% | 136 | 1.6 | 0.11 | 9 | 14 | 14 | 2.6 | 0.05 | 2 | 7 | 9 |
| 5% | 68 | 1.0 | 0.11 | 8 | 14 | 14 | 1.9 | 0.03 | 6 | 6 | 6 |

Table 5.3 summaries the results, where %P and #P are respectively the percentage and the number of most sparse predicates as goal predicates, Time is the system time for model learning and link prediction (in hours, disregarding the times for computing source embeddings which could be done offline), and H@1, H@3, and H@10 are the Hits@1, Hits@3, and Hits@10 scores in percentage.

On such sparse predicates, TRL again outperformed RLvLR regarding both prediction accuracy and time efficiency. While HOLE and TransE showed better accuracy on 20% most sparse predicates, as the sparsity increases, TRL outperforms them on Hits@3 and Hits@1 accuracy, which means top ranked predictions from TRL are more likely to be correct. Note that for top 5% sparse predicates, TRL could hit the target with its top one prediction (i.e., Hits@1) for 8% of the cases, whereas the other systems failed in all cases. Note that also, while HOLE showed outstanding accuracy regarding Hits@10 for all cases, the learning time it took was at least 4 times more than that of TRL. The time consumption for TransE and HOLE were relatively constant, because unlike TRL or RLvLR, neither of them can learn models for specific goal predicates.

To analyse the contributions of the two similarity measures presented in Section 5.3.2, we tested three configurations of TRL on 10% most sparse predicates: with linear transformation (LT) only, with translation with argument embeddings (TAE) only, and with both. Other settings are as before, and Table 5.4 reports the prediction accuracy of learnt rules.
Using both LT and TAE showed the best prediction accuracy, which justifies our default configuration of combining them. The performance is suboptimal when using either LT or TAE alone, while the performance of TAE alone is slightly better than that of LT alone. This demonstrates the usefulness of argument embeddings in measuring similarity.

### 5.5 Related Works

We are unaware of any approach on embedding-based transfer rule learning for KGs. Transfer learning on relational data has been intensively studied in the literature, examples include [51, 76]. These approaches transfer logical probabilistic models such as Markov Logic Network (MLN), which can be expressed as a form of rule languages, from source domain to address the issue of sparse data in the target domain. However, these transfer learners assume a known source domain that is similar enough to the target domain for transfer, which may work for specific domains but not for KGs with general knowledge. Our system, on the other hand, can explore the existing KGs and retrieve source tasks.

In [77], the authors proposed an embedding-based transfer learning approach for KG completion, but they do not consider rule learning. Instead of transferring rules as in our work, the proposed approach enhances the performance of current embedding-based KG completion system by transferring model parameters. Other embedding-based systems for KG completion [18], such as RESCAL, HOLE [15], and TransE [16], do not use transfer learning. These embedding-based systems (including [77]) cannot handle the new KGs scenario as they require rich data to train the embedding-based models, which are directly used for prediction.
Ontology alignment, which concerns the alignment between elements from distinct ontologies, is also related to our predicate mapping method, and some ontology alignment techniques have been adapted to KGs especially for KGs of different human languages [75, 78]. Yet our predicate mapping method is different from ontology alignment in the notion of similarity. Ontology alignment considers two elements to be similar if they represent the same objects in the real world, such as “cat” in English and in Japanese. On the other hand, our predicate mapping concerns structural similarity in the context of their respective KGs.

5.6 Conclusion

In this chapter, we proposed an approach to mining first-order rules over sparse data in KGs, and we achieved this through embedding-based transfer learning. In contrast to existing transfer rule learning approaches, using embedding representation of predicates enables us to measure structural similarity between goal predicates of tasks and retrieve source tasks that are similar to the target task. Also, using embeddings allows us to map the predicates between source tasks and the target task. Our embedding-based measures were shown to be effective for transfer rule learning and helped to reduce negative transfer. Finally, we evaluated our system TRL on widely used benchmarks in rule mining and link prediction. Our experimental results demonstrate that TRL outperformed AMIE+ and RLvLR, two state-of-the-art rule learners, in terms of both the number of mined quality rules and accuracy in link prediction. While our system is not specifically design for link prediction, we demonstrated the predictive power of our mined rules through a comparison with link prediction systems TransE and HOLE. Again, TRL outperformed them in terms of prediction accuracy and time efficient on sparse predicates.
Chapter 6

Learning Probabilistic Rules via Transfer Learning from KBs

In this chapter, we consider learning probabilistic logical rules instead of learning logical rules. The rules which we learned in the previous chapters are augmented with some quality degrees like standard confidence and head coverage. Although we use these qualities to filter out some rules and to infer the confidence degree of new facts which we drive, these augmented qualities do not represent the probabilistic feature of rules. To learn a model which has a well-defined probabilistic feature we use LPAD as the formalism of the output models in this chapter.

While leaning the structure of logical rules is a challenging task due to the vast space of candidate rules, learning a set of probabilistic rules (i.e. LPADs) in order to their structures and parameters are fitted to the given data is a quite more challenging task. It is also an issue even for state-of-the-art learning system, SLIPCOVER [13] which learns parameter and structure of LPADs.

In this chapter, we present a new algorithm T-LPAD for structure learning of LPADs by employing transfer learning. The transferred knowledge aids the learning process to restrict the space of candidate rules and demand less training data. A prototype of the system is implemented, and our experimental results show that T-LPAD outperforms SLIPCOVER (and SLIPCASE) for most benchmarks used
in related systems. Since the data which we consider in this chapter is different from the data which we considered in the previous chapters (KGs) as we explain it in the following, we could not use RLvLR and TRL in this task.

To compare with existing probabilistic rule learners, we consider a more general kind of data. On the contrary of previous chapters which we consider KG data, in this chapter we consider facts with arbitrary arity. KG is a special kind of knowledge base which each fact has binary arity. Moreover, on the contrary to KGs which contain only positive facts, in this chapter, the KBs include positive and negative facts. Consider the more general format of input data comes with a cost. We cannot use efficient graph manipulation tools including efficient inference method and embedding representation to carry out the inferencing and learning tasks.

This chapter resulted in the following publication:


6.1 Introduction

In logic programming, researchers have realized the importance of modeling uncertainty for a long time and thus various types of probabilistic logic programs have been proposed and studied. Consequently, significant attention has been paid to the issue of learning probabilistic logic programs. Recently, several approaches have been proposed for parameter learning (i.e., for a candidate logic program, the task is to determine probability degrees for certain objects such as rules and/or atoms). For instance, the PRISM [60] is well known for its distribution semantics. LeProbLog [79] is based the technique of gradient descent while LFI-ProbLog [80] and EMBLEM [65] adopt an Expectation Maximization approach in which the expectations are computed directly using binary decision diagrams (BDDs). In
many realistic applications, we need also to learn candidate logic programs as well as learning parameters. This paradigm of probabilistic logic program learning is referred to as structure learning. Obviously, this task is much more difficult than parameter learning only since in the structure learning the parameters should be learned iteratively to assess the quality of discovered potential structures. ProbLog is proposed by De Raedt et al [81] for learning the structure of (probabilistic) logic programs while SEM-CP-logic is developed for learning ground LPAD programs. More recently, based on a new beam search, SLIPCASE [64] is developed for learning LPAD programs. SLIPCASE can learn general LPADs including non-ground programs. An improved version of SLIPCASE, named SLIPCOVER, is described in [13]. In learning algorithms for both SLIPCASE and SLIPCOVER, beam search is performed in the space of LPADs using the log likelihood of training data as the guiding heuristics and theory refinements are achieved using EMBLEM. However, there is still significant room for improving the efficiency of these algorithms. Especially, in these algorithms, the search space of LPADs is still very large and thus it would be useful to reduce the search space for structure learning algorithms using information from another application domain that shares a certain similarity with the domain of interest.

Human beings can make analogy across different domains by determining the structural similarities even in seemingly irrelevant domains, for example, we can easily understand the analogy between the domain of movie information and the domain of academic information. Even though the movie domain has nothing in common with the academic domain, we can still make the analogy based on certain similarity. For example, the predicate “Movie(title, person)” with two arguments of types “title” and “person” is similar to the predicate “Publication(person, title)” in the academic domain; the predicate “Director(person)” is similar to “Professor(person)”; the predicate “Actor(person)” is similar to “Student(person)”.

As we discussed in previous Chapters 2 and 5, reusing knowledge across different domains for learning has been actively pursued in transfer learning.
In this chapter, we apply transfer learning in learning LPADs and describe a new algorithm T-LPAD for structure learning of LPADs (here ‘T’ in T-LPAD is for ‘Transfer learning’). Specifically, suppose that our task is to produce a set of LPAD rules for a domain of interest (i.e., target domain) based on a given training (relational) dataset. At the same time, we are given a set of LPAD rules in another domain (i.e., source domain) that is independent of the target domain but shares a kind of structural similarity with the target domain. By employing a technique from ontology matching [82], we are able to figure out a similarity degree for a pair of predicates in the source domain and the target domain. Using these similarity degrees, we construct rules in the target domain based on those rules in the source domain. This will allow us to narrow down the search space of constructing candidate rules in structure learning of LPADs. Note that these rules are standard (disjunctive) rules but not LPAD rules since no probability degrees are assigned to head atoms. For example, from the academic domain, we can produce a rule 
\[ \text{Director}(\text{person}); \text{Starring}(\text{person}) \leftarrow \text{Movie}(\text{title}, \text{person}), \text{Bigname}(\text{person}) \].

In the next stage, we figure out the best probability for each predicate in the rule head for each disjunctive rule using existing parameter learning algorithms such as EMBLEM. For instance, we can come up with an LPAD rule like 
\[ \text{Director}(\text{person}) : 0.4); (\text{Starring}(\text{person}) : 0.6) \leftarrow \text{Movie}(\text{title}, \text{person}), \text{Bigname}(\text{person}) \].

We have developed a prototype implementation for T-LPAD algorithm and conducted experiments on four benchmarks IMDB, UW-CSE, WebKB, and Twitter, which are widely used and publicly available [13, 51]. The experimental results show that T-LPAD outperforms SLIPCOVER (and SLIPCASE), two latest algorithms for structure learning of LPADs.

In the rest of the chapter, our new learning algorithm T-LPAD is described through a running example in Section 6.2, some experimental results are reported in Section 6.3. Finally, we discuss the relation of our approach to some other approaches in Section tlpad:sec:rw and we conclude the work in Section 6.5.
6.2 An Overview of Our Approach

In this section, we present a new approach to learning LPAD rules through transferring knowledge across domains. We will first provide a sketch of our learning algorithm T-LPAD and then explain further technical details in the sections through a running example.

Suppose that we have two application domains in hand. One is the source domain, which is relatively well understood, and the other is the target domain in which we want to learn new knowledge in the form of LPAD rules. More formally, the source domain has a set of relational data and a set of LPAD rules that have previously obtained. The target domain has only a set of relational data but does not have any rules. The transfer learning task for the given domains is that, given a predicate in the target domain, we construct a set of LPAD rules for the target domain that are compatible with the (training) data in the target domain.

In our T-LPAD algorithm, for each domain we first construct a language bias called predicate description (PD), which contains information about predicate arity, predicate arguments, and their types. A PD can also be represented as a matrix (PD matrix). Based on two PD matrices for the source domain and the target domain, for each target predicate $P$, we are able to determine a set $S(P)$ of source predicates that are structurally similar to the given target predicate. In this way, we can construct a set of LPAD rules for the target domain from an LPAD rules in the source domain, by replacing each source predicate with a target predicate that is structurally similar to it.

In the next two sections, we will explain our method using the following example. 

Example 4 (Running Example). The source domain and the target domain are Academic and Movie, respectively. The academic covers knowledge about people in an academy department (i.e. students and professors) and their relationships. The movie domain contains information about movies, their directors, and the actors of the movies. The predicate descriptions for a Academic domain and the Movie domain are specified as follows.
Academic PD: Advisedby(person, person), Professor(person), Student(person), Publication(person, title), Position(person, +pos)

Movie PD: Workunder(person, person), Actor(person), Movie(title, person), Director(person)

Academic domain contains one rule Professor(x) : 0.2 ← Publication(x, y).

Based on the above information, we want to learn rules for Movie about Director (i.e., rules with Director in the head).

### 6.2.1 PD Graph and Similarity Matrix

To extract similarity between the predicates in the source domain and those in the target domain, we make use of the PDs, which describes the predicates in the domain, their arities, the types of their arguments, and how each argument appears in rules (i.e., as variables or constants).

The predicate description for a domain can be conveniently represented as a graph called the PD graph for the domain.

**Definition 1.** Given a domain $D$, its PD graph, denoted $G_D$, is defined as follows:

1. The vertex of $G_D$ is either a predicate $P/m$ with $m$ denoting its arity, or a type $T$ or $+T$ with + denoting it is a constant type.

2. If the $k$-th argument of a predicate $P$ is the type $T$, then there is a directed edge labelled $\#k$ from vertex $P/m$ to vertex $T$.

The PD graphs for those two domains in Example 4 are shown in Figure 6.1.

We want to construct a similarity matrix using PD graphs, in which each pair of source and target predicates is assigned a similarity degree as a real number. To achieve this, we adapt existing graph matching techniques which can extract similarity degree between the vertices of two graphs based on their structural
similarity. Our PD graphs can be conveniently represented as RDF triples [82], where an RDF triple represents one edge or a labelled vertex of a PD graph.

Our RDF encoding contains three kinds of triples:

1. The predicate-type triples $(P, \#k, T)$ expressing that the $k$th argument of predicate $P$ has the type $T$. Each triple of this type encodes one edge in the PD graph.

2. The predicate triples $(P, n, \text{predicate})$, encoding that vertex $P$ represents a predicate with arity $n$.

3. The type triples which has form $(T, \text{cons/var}, \text{type})$, encoding that vertex $T$ represents a constant or variable type.

Encoding PD graphs as RDF triples allows us to adapt the RDF matcher GMO [82] to extract a graph matching matrix, denoted $GM$, which can be seen as a function that maps each pair of predicates to a real number. We use $GM$ as a basis to construct the similarity matrix between source and target predicates.
Chapter 6. \textit{T-LPAD}

The graph matching matrix $GM$ is completely based on structural similarity, which does not necessarily reflect the different labels in graphs, for example, whether a vertex is a binary predicate or a constant type. Hence, we need to refine $GM$ to reflect the different reasons for two predicates to be considered similar, i.e., due to the same arity or similar argument types. We compute a label matrix $LM$ to capture the similarity due to different labels, by comparing labels using an adapted string matching method \cite{82}. We refine $GM$ by compositing $GM$ and $LM$ linearly with parameter $\beta$ into a refined matrix $RM$ as the following.

$$RM = \beta \ast LM + (1 - \beta)GM$$

(6.1)

The refined matrix $RM$ can be divided into two matrices, $RM_t$ and $RM_p$ which present the similarity between types and predicates respectively. To obtain the final similarity matrix $SM$, we further refine $RM_p$ through $RM_t$ according to the argument types of predicates. For this refinement step, we use the linear combination (6.2). Again, each matrix can be seen as a function mapping each pair of predicates (or types) to a real number. For a predicate $P$, $T^P$ is the set of all the types of the arguments of $P$, and $T^P_i$ is the type of $i$th argument of $P$. To define $SM$, for a pair of predicates $P$ and $Q$,

$$SM(P, Q) = (1 - \alpha) \ast RM_p(P, Q) + \alpha \ast \frac{1}{m} \ast \sum_{i,j} \max(RM_t(T^P_i, T^Q_j))$$

(6.2)

where $m = \max(|T^P|, |T^Q|)$ and $n = \min(|T^P|, |T^Q|)$. The parameter $\alpha$ determines the degree how much the similarity degrees in $RM_p$ are refine by the similarity between types in $RM_t$. In particular, the second part of Equation (6.2) aggregates the similarities degrees of types for the two predicates $P$ and $Q$. For example, to compute the similarity degree between two predicates \textit{Student(person)} and \textit{Movie(title,person)}, we have $m = 2$ and $n = 1$. Taking $\alpha = 0.5$, suppose $RM_p(\text{Student, Movie}) = 0.3$, $RM_t(\text{person, title}) = 0.3$, and $RM_t(\text{person, person}) = 1$, we can compute $SM(\text{Student, Movie}) = 0.5 \times 0.3 + 0.5 \times 0.5 \times \sum_i \max(0.3, 1) = 0.4$. 


We summarize the computation of the similarity matrix in Algorithm 4.

**Algorithm 4 Compute the similarity matrix**

**Input:** source PD $S$ and target PD $T$

**Output:** similarity matrix $SM$

1. Encode $S$ and $T$ into RDF documents
2. $GM \leftarrow GMO^*(S,T)$ \hspace{1cm} $\triangleright$ GMO$^*$ is our adapted GMO method
3. $LM \leftarrow LabelStringMatch(S, T)$
4. $RM \leftarrow Refine(GM, LM)$ \hspace{1cm} $\triangleright$ Refine is defined in Equation (6.1)
5. $SM \leftarrow FurtherRefine(RM_p, RM_t)$ \hspace{1cm} $\triangleright$ FurtherRefine is defined in Equation (6.2) \hspace{1cm} **return** $SM$

**Example 5 (Cont’d Example 4).** Taking as input the PDs of Academy and Movie domains, Algorithm 4 outputs the similarity matrix $SM_{Academy,Movie}$ as follows:

$$SM_{Academy,Movie} = \begin{bmatrix}
\text{Advised by} & 0.85 & 0.5 & 0.5 & 0.7 \\
\text{Student} & 0.5 & 0.8 & 0.8 & 0.4 \\
\text{Professor} & 0.5 & 0.8 & 0.8 & 0.4 \\
\text{Position} & 0.7 & 0.4 & 0.4 & 0.8 \\
\text{Publication} & 0.7 & 0.4 & 0.4 & 0.8
\end{bmatrix}$$

### 6.2.2 Rule Construction

In what follows, we show how rules can be constructed for the target domain based on the similarity matrix. In contrast to existing approaches, which transfer second order template, we use rule templates of the form $P_1; \ldots; P_m \leftarrow Q_1, \ldots, Q_n$ where $P_i$ and $Q_j$ are predicates (without arguments). We first show how our method works for rules with a single-atom head, and then extend the approach to rules with multiple-atom heads. For the convenience of discussion, we omit the weight of LPAD rules in our examples.

Let $SM$ be the similarity matrix obtained previously, for each predicate $P$ in the target (or source) domain, let $S(P)$ (resp., $T(P)$) be the set of source (resp., target) predicates that have the highest similarity degree to $P$ in $SM$. To construct target rules with goal predicate $G$, our rule construction method consists three steps.
In the first step, for each source predicate \( P \in S(G) \) and each rule \( r \) of the form \( P(\vec{t}) \leftarrow Q_1(\vec{t}_1), \ldots, Q_n(\vec{t}_n) \) in the source domain, we obtain a set \( R_0(r) \) of rule templates in the target of the form \( G \leftarrow Q'_1, \ldots, Q'_n \), where \( Q'_i \in T(Q_i) \). For example, suppose \( r \) is \( A(x,y) \leftarrow B(y), C(x,z) \), \( T(B) = \{B', B''\} \), and \( T(C) = \{C'\} \). Then \( R_0(r) \) contains two rule templates: \( G \leftarrow B', C' \) and \( G \leftarrow B'', C' \).

In the second step, we assign variables and constants as arguments to the predicates in \( R_0(r) \). If a predicate has an argument type that is a constant type, we construct rules by assigning all possible constants of that type. Assignment of variable is less straightforward. We first assign a distinct variable to each argument which is not a constant, and then unify certain variables based on the variable sharing information in \( r \), which we capture using variable sharing constraints. A variable sharing constraint in \( r \) is of the form \( x(P_{m_1}, \ldots, P_{m_k}, Q_{n_1}, \ldots, Q_{n_l}) \), where \( P_{m_i} \)'s and \( Q_{n_l} \)'s are the head and body predicates that have \( x \) as an argument. In the previous example, the variable sharing constraints in \( r \) are \( x(A, C) \) and \( y(A, B) \). Based on variable sharing constraints, we unify the assigned variable according to the following three conditions: (1) Two variable can unify only if they have the same type. (2) If two predicates share a variable in \( r \), the corresponding predicates in the rule template will have one variable unified whenever possible. (3) Unification is performed only when it is required by condition (1). In this way, we obtain a set of rules \( R_1(r) \) from \( R_0(r) \) through assignment. In the above example, suppose arities of \( G, B', B'' \) and \( C' \) are respectively 2, 1, 2 and 1, \( R_1(r) \) may contain some of the following rules (regardless of types): \( G(x,y) \leftarrow B'(x), C'(x), G(x,y) \leftarrow B''(x,z), C'(y), G(x,y) \leftarrow B''(y,z), C'(x), G(x,y) \leftarrow B''(z,x), C'(y), \) and \( G(x,y) \leftarrow B''(z,y), C'(x) \).

Example 6 (Cont’d Example 5). To learn a rule about \textit{Director} in the Movie domain, we obtain from our similarity matrix \( S(\text{Director}) = \{\text{Professor}, \text{Student}\} \) and source rule \( r : \text{Professor}(x) : 0.2 \leftarrow \text{Publication}(x,y) \). Also, we have \( T(\text{Publication}) = \{\text{Movie}\} \). In this case we get one rule template in \( R_0(r) \), that is \( \text{Director} \leftarrow \text{Movie} \). After assigning arguments based on the single variable sharing constraint in \( r \), that is \( x(\text{Professor}, \text{Publication}) \), we obtain one rule in \( R_1(r) \), that is \( \text{Director}(x) \leftarrow \text{Movie}(y,x) \).
T-LPAD handles rules with multiple-atom head in a similar way. We illustrate this using an example. Suppose the source rule \( r \) is \( A(x, y); B(x, y) \leftarrow C(x), D(y) \) and from the similarity matrix we obtain \( T(A) = \{ A', B' \} \), \( T(B) = \{ C' \} \), \( T(C) = \{ D' \} \), and \( T(D) = \{ E' \} \), then \( R_0(r) \) consists of the following two rule templates: 

\( A'; C' \leftarrow D', E' \) and \( B'; C' \leftarrow D', E' \). The assignment of argument works exactly as for rules with single-atom heads, and hence is not repeated here.

The rules constructed from the above two steps do not have weights in their head. Hence, in the third step, we generate weights for them. We initialise the weights of head atoms evenly, through dividing 1 by the number of head atoms. For example, in the cases with one head atom, noting that there is an implicit null atom, we assign 0.5 to each of these two. Then, we feed these rules to the parameter learner EMBLEM [65] to induce weights. Finally, rules with probability degrees below a threshold are eliminated from the candidates.

### 6.3 Experiments

In our experiments, we compare our T-LPAD with the state-of-the-art LPAD learner SLIPCOVER [13]. T-LPAD learns rules through transferring knowledge from an unrelated domain, whereas SLIPCOVER learns rules directly from facts. We try to address the following three questions:

- Can T-LPAD learn rules with high accuracy comparable to what SLIPCOVER can learn?

- Can T-LPAD learn such rules with (much) smaller amount of data than SLIPCOVER?

- Does T-LPAD learn such rules (much) faster than SLIPCOVER?

We used four domains IMDB, UW-CSE, WebKB, and Twitter, which are widely used and publicly available [13, 51]. Part of the IMDB domain about movies and the UW-CSE domain about academics have been used in the running
example. The WebKB domain describes web pages from the computer science departments of four universities, and the Twitter domain contains tweets about Belgian soccer matches. As with existing approach, the facts in each domain are divided into smaller sets called folds [13, 51]. To evaluate the performance of learners on small amount of data, we further divide the existing folds into smaller ones. In our experiment, IMDB has 10 folds of facts, UW-CSE has 10 folds, WebKB has 8 folds, and Twitter has 8 folds.

To perform transfer learning, we paired up the domains into two pairs: IMDB with UW-CSE and WebKB with Twitter; The two domains in each pair serve as source and target respectively in one round of evaluation, and swap roles in another round. The goal predicate (i.e., that occurring in the head of rules to learn) for IMDB is Workunder, for UW-CSE is Advisedby, for WebKB is Coursepage, and for Twitter is Accountfan. These predicates were picked following the existing literature. Rules in each source domain either came with the domain or learnt from the facts.

For all learning tasks, we evaluated T-LPAD under the same configuration: graph matching steps $\leq 30$ and convergence threshold $10E-9$, $\alpha = 0.5$, and $\beta = 0.3$. We used EMBLEM [65] to learn weights for the candidate rules. For SLIPCOVER, we adopted the parameters recommended by [13] for each domain. For a fair comparison, we disabled domain-dependent heuristics like “Lookahead” in SLIPCOVER. All the evaluation was conducted on a PC with 8 G RAM and corei5 CPU.

To evaluate and compare T-LPAD and SLIPCOVER, we adopted the standard measurement test set Log Likelihood (LL) and Area Under the Receiver Operating Characteristic Curve (AUROC) for measuring accuracy of the learnt rules [13]. LL directly measures the quality of the probability estimates produced, and the advantage of the AUROC is that it is insensitive to the large number of true negatives and it used both sensitivity and specificity for all possible thresholds.

Table 6.1 shows the LL and AUCROC for SLIPCOVER and T-LPAD with varying amounts of training data. In particular, “Number of folds” refer to the number of folds used for training the learners (T-LPAD uses training data only as inputs
of EMBLEM), whereas the remaining data will be used for evaluating the learnt rules. The results show that regarding to the accuracy of learnt rules, T-LPAD was comparable to SLIPCOVER, and in several cases outperformed SLIPCOVER. The result was surprising as T-LPAD uses limited knowledge from an unrelated domain; yet on the other hand, it suggests the tight connection between the PD and rule structure of a domain. Through graph matching and the similarity matrix, T-LPAD is able to exploit such connection to a large extent and extract critical information for rule construction. The advantage of T-LPAD become obvious when the training data is reduced. In particular, with 1 fold of training data, SLIPCOVER suffered from insufficient training data whereas the performance of T-LPAD was reasonably stable. This is due to the fact that T-LPAD learns by transferring knowledge from another domain and hence is not data hungry.

We also measured the times needed for training with 6 folds, as shown in Table 6.2. All times are in minutes. From the results, it is clear that T-LPAD learns faster than SLIPCOVER when more training data are involved.

<table>
<thead>
<tr>
<th></th>
<th>SLIPCOVER</th>
<th>T-LPAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW-CSE→IMDB</td>
<td>-891.1</td>
<td>0.1</td>
</tr>
<tr>
<td>IMDB→UW-CSE</td>
<td>-37.63</td>
<td>0.1</td>
</tr>
<tr>
<td>Twitter→WebKB</td>
<td>5.25</td>
<td>3.15</td>
</tr>
<tr>
<td>WebKB→Twitter</td>
<td>2.1</td>
<td>0.4</td>
</tr>
</tbody>
</table>

**Table 6.2:** Training times with 6 folds training data.

To conclude, with sufficient training data, the accuracy of T-LPAD is comparable to SLIPCOVER but T-LPAD is faster than SLIPCOVER. In the case where only a small amount of training data is available, T-LPAD outperforms SLIPCOVER.
regarding to the quality of rules learnt. Hence, T-LPAD is more robust than SLIPCOVER in terms of small training data.

### 6.4 Related Works

T-LPAD belongs to the class of deep transfer learning methods which are capable of generalizing knowledge cross distinct domains. Conceptually, the closest transfer learning approaches to T-LPAD are TAMAR [52], DTM [50], TODTLER [51], and transfer learning with type matching [53] which perform deep transfer in the context of Markov Logic Networks. One difference between our approach and these existing ones is about what knowledge is transferred. In particular, T-LPAD produces a similarity matrix, which associates the predicates of two domains based on their similarity in the predicate descriptions.

In [51, 52], the knowledge from source domain is transferred through an intermediate knowledge language called second-order templates. Recall our running example, from the source rule $\text{Professor}(x) : 0.2 \leftarrow \text{Publication}(x, y)$, a second order template $X(x) \leftarrow Y(x, y)$ can be obtained. Assume we want to learn a rule about $\text{Director}$ in the Movie domain, by initialising the second template, a desired rule $\text{Director}(x) \leftarrow \text{Movie}(y, x)$ cannot be obtained from the second order template. This is due to the fact that second order templates cannot capture the similarity between predicates beyond their arity and the order of their arguments. Yet from the PDs and especially the graphs in Figure 6.1, it is clear that predicate $\text{Movie}$ in the Movie domain is most similar to the $\text{Publication}$ in the Academy domain.

Existing approach [53] learns by transferring, instead of second order rule templates, type sharing knowledge among predicates. In particular, $\text{Professor}$ and $\text{Publication}$ share one type $\text{person}$ and one variable $x$ in the rule $\text{Professor}(x) : 0.2 \leftarrow \text{Publication}(x, y)$ of Academy. Rules $\text{Director}(x) \leftarrow \text{Workunder}(x, y)$ and $\text{Director}(x) \leftarrow \text{Actor}(x)$ can be constructed for Movie in [53] as candidates, as $\text{Director}$ shares one type $\text{person}$ and one variable $x$ with $\text{Workunder}$ in the former rule and $\text{Actor}$ in the latter one. Yet again, these candidates are counter-intuitive.
Matching types allows more flexibility than second order template (e.g., in the above example, Actor can be in the place of Publication), but it does not take into account the similarity between predicates.

Another distinguishing feature of our approach is that our system does not require fact level data for structure learning. All the other approach require a refinement procedure to eliminate inaccurate rules, which is often data hungry. Also, the computational cost of such refinement is often high compared to the process of similarity matrix and rule construction in our case.

6.5 Conclusion

We have proposed an algorithm T-LPAD for LPAD structure learning using transfer learning, based on the similarity between two independent problem domains. Our algorithm identifies such similarity by matrix matching, and uses it to guide the candidate rule crafting for target domain in the presence of rules in the source domain. We have implemented the T-LPAD algorithm and conducted experiments for Web and social network domains. Our experimental results show that T-LPAD outperforms SLIPCOVER, a major structure learning algorithm for LPADs.
Chapter 7

Conclusion and Future Work

In this thesis, we have illustrated the outstanding benefits of using ML approaches along with KR approaches in learning predictive models in the form of logical rules from knowledge bases in different scenarios. The contributions of this dissertation can be characterized by four major directions: (1) We developed a novel approach to learn rules from vast KGs via adopting representation learning techniques. We also developed a method for applying the learned rules to complete the KGs. (2) We proposed an approach to learn temporal rules from a stream of KGs. In our proposed method the learning and reasoning from/through the KG streams can be done concurrently. (3) We proposed a novel transfer rule learning approach to learn rules from sparse KGs by deploying the embedding representation of KGs. (4) We developed a method to learn the structure of probabilistic rules from the KBs through transfer learning and matrices matching.

Taken together, the contributions in this dissertation have led to an advancement in KG logical modeling, an essential task to maintain and manage KG which is an increasingly popular kind of data storage. We believe that our research is a step toward the broader use of logic-based modeling in the KG-related research community. We also hope the proposed human-understandable learning and reasoning approaches contribute to the development of more reliable and risk-free ML methods in the future.
Chapter 7. Conclusion and Future Work

7.1 Results Summary

While massive knowledge graphs have proven valuable for a variety of tasks, they are far from complete. The aims of this dissertation are to develop approaches that enable Learning a logical model from such KGs under different learning scenarios and applying the learned models to the KG to infer the new facts. In this dissertation, we have presented four methods for learning rules from knowledge bases as follows.

In Chapter 3, we have proposed an approach to extract closed path rules (a simple while expressive fragment of first-order rules) from Knowledge Graphs. RLvLR deploys the embeddings of predicates and arguments as a heuristic to explore the space of hypotheses. It also uses a novel target oriented sampling. In this procedure, it forms a sampled KG which is far smaller than the original KG yet contains the necessary information regarding the rule learning task. Both the proposed heuristic search and sampling method contribute to the scalability of RLvLR in learning complex quality rules from KG with over 10 million facts in a reasonable time.

Our experimental results have shown that RLvLR outperforms AMIE+ regarding rule quality and efficiency. For link prediction, RLvLR outperforms Neural LP regarding time efficiency and accuracy.

The performance of RLvLR has shown it can learn more complex (longer) rules from KGs with a shorter runtime in comparison with the state-of-the-art rule learner (AMIE+). We also assessed the quality of the learned rules not only by the standard quality measures of rules (i.e., head coverage and standard confidence) but also by deploying them to carry out the link prediction task. The accuracy of the learned rules is evidence for the usefulness of RLvLR-mined rules. Consequently, RLvLR can be an alternative for learning and reasoning over the large-scale KGs which are increasingly popular nowadays.

In Chapter 4, we considered the stream of KGs. We have proposed a method to learn temporal rules from KG streams and to apply the learned rules to the stream
in order to predict the upcoming events. The temporal rules are helpful to carry out the link prediction in the KG stream particularly when a restricted range of historical data is available to the system.

Our approach adopts the static rule quality measures to handle the dynamic nature of data in the KG stream. We implemented our method, named StreamLearner, based on our static rule learner (RLvLR).

The experimental results have shown that StreamLearner outperformed some state-of-the-art embedding-based learners, HOLE and TransE, in the link prediction task. Since StreamLearner deploys rules to formalize the learned knowledge from KG streams, it can inherently scale up to handle large-scale stream data.

In Chapter 5, we focused on the sparsity in KGs. In this chapter, we have proposed an approach to learn first-order rules over sparse data in KGs. To do so, we deploy representation learning methods along with transfer learning. Our proposed method, named TRL, retrieves the most similar source task from provided KGs which are accompanied with rules. Using embedding presentation of predicates enables TRL to measure structural similarity to retrieve the most similar source tasks as well as to map the predicates between source tasks and the target task. The proposed embedding-based similarity measures were proven to be beneficial for transfer rule learning and help to decrease negative transfer.

The experimental results have shown that TRL outperforms AMIE+ and RLvLR, state-of-the-art rule learners, in terms of the number of mined rules and the accuracy of link prediction in the sparse KGs. Although the goal of TRL is not specifically addressing the link prediction task, the experimental results showed TRL outperformed link prediction systems (TransE and HOLE) in terms of prediction accuracy and time efficiency on sparse predicates.

We considered two scenarios in which the learner should learn from sparse data; a part of the KG is sparse, and the whole KG is sparse. Regarding the first scenario, we can apply the TRL to the large-scale KG which has limited information regarding a number of critical predicates for us. This is a common situation since
the demanding of a predictive model regarding a predicate is more essential when less related facts are available. TRL also can be deployed in the KGs which are generally sparse. We face this scenario due to many situations such as when the acquiring of data is expensive.

In Chapter 6, we have proposed a novel method, named T-LPAD, which deploys LPAD as the formalism of its learned rules. Besides, it considers a more general form of data as input. While the KG contains the facts with binary arity, T-LPAD handles facts with arbitrary arity. T-LPAD learns LPAD structure by using transfer learning, based on the similarity between two independent problem domains. It determines such similarity according to the type information of a domain’s predicates. It uses the type information through matrix matching. The transferred knowledge guides the candidate rule crafting for the target domain in the presence of rules in the source domain. The experiments have shown that T-LPAD outperformed SLIPCOVER, a major structure learning algorithm for LPADs.

The more expressive kind of learned rules (i.e., LPAD has disjunction in its head and the probabilistic feature) and more general form of input data come at a cost, efficiency. Thus, the size of data which T-LPAD can handle is incomparable with the size of input KGs which are considered in the previous chapters.

### 7.2 Discussion and Future Work

The growth of vast KBs provided opportunities for a knowledge managing framework to manipulate the data from different perspectives including reasoning and learning. While the statistical approaches (e.g. Representation Learning and Neural Networks) have shown excellent performance in the learning task, the lack of transparency of their obtained model is an issue. The learners that can learn logical human-understandable models address this issue. We believe in the future learning systems are expected to answer ‘why?’ question besides ‘what?’ question. To answer the why question, the learner should provide some additional
information regarding its answer. The additional information helps the human to understand why the answer is likely to be true. We believe the proposed learning systems which learn logical models from data and use the logical model to predict the missing facts is an important step toward such learning systems.

In the following, we present an overview of the potential directions of future research regarding the learning logical models from KBs.

**Extending the expensiveness of logical rules** While in a number of our approaches (i.e., RLvLR and StreamLearner), we use CP rules as the formalism of gained knowledge, considering rules with extended formalism is a promising direction. The first extension is considering a fragment of first-order rules with less restrictive syntax like Horn rules. Finding a more expressive language bias that does not harm the efficiency of proposed learning and reasoning methods is a challenge.

The second interesting extension of CP rules is learning rules with constants. The proposed learning methods learn rules which are completely abstract (constant-free). While learning rules with constants aids the learner to capture the existing patterns in data more precisely, it greatly extends the space of candidate rules. Lastly, the proposed methods do not learn rules with negative atoms in the body. Since in the KGs under OWA, we do not have any explicit negative fact, we plan to consider negation as failure as it is defined in the Logic Program. In this setting, a negative fact \( \text{not}b \) becomes true if all efforts to prove \( b \) fail in finite time.

**Parallel implementation of RLvLR** RLvLR explores the space of candidate rules heuristically based on the embedding presentation of predicates and entities. We plan to improve the efficiency of the system by performing part of computations parallelly. The current algorithm learns rules with different target predicates independently, which is useful for developing a parallel algorithm. The optimum parallel implementation can be done by deploying the MapReduce approach.
Extending StreamLearner by considering more properties of facts In StreamLearner we consider an atomic time point as the time of an event. By considering more attributes of an event, we plan to develop a learning system which captures the relations in the data more precisely.

The available properties which can be included are the duration of an event or the occurrence place of the event. In this case, the learned rules are sensitive to all these properties along with the time. Although the rules with more details reflect the data more accurately, they impose further complexity to the learning and reasoning process. Thus, adopting the proposed system to handle this task efficiently is an issue.

Iterative transfer rule learner We plan to extend our transfer rule learner by deploying an iterated learning framework. In this setting, the rules learned via transfer can be applied to sparse data and infer new facts. In the next iteration, the new rules are learned from the enriched data (original data and the new inferred facts). This semi-supervised learning setting elevates the sparsity of data by enriching the sparse KG in each iteration incrementally.
Bibliography


