THE UNIVERSITY *of* EDINBURGH

Schema Aware Knowledge Graph Completions

Fangrong Wang

Doctor of Philosophy

THE UNIVERSITY OF EDINBURGH

2023

Abstract

Knowledge Graph Completion (KGC) aims to complete the structure of knowledge graphs by predicting the missing entities or relationships in them and mining unknown facts over the relational triples of the form $\langle h, r, t \rangle$, where h is the head entity, r the relation and t the tail entity. Recent success of knowledge graphs (KG) has spurred widespread interests in methods for the problem of KGC. However, efforts to understand the quality of the candidate triples from these methods, in particular from the schema aspect, have been limited. In fact, most existing Knowledge Graph completion methods do not guarantee that the expanded Knowledge Graphs are consistent with the schema of the initial Knowledge Graph. As a result, while existing KGC approaches perform well in completing the graph under the rank based evaluation metrics such as Hit@N and Mean Reciprocal Rank, they often struggle to ensure the consistency of newly generated triples. Therefore, an approach that tries to balance completeness and consistency is needed.

Existing KGC research often uses the silver standard method [1] to measure the performance of Knowledge Graph completion approaches, assuming that the KG itself is already of reasonable quality. In the silver standard method, some existing links in the data sub-graph (ABox) are removed for testing if triple producers can help to recover the missing links. Our experiments bring the silver standard method into question. They show that only 89.9% of triples are consistent with the original NELL-995 dataset [2] schema and satisfy the domain and range constraints. The corresponding ratios for the DBpedia Politics subset (DBped-P) [3] are 99.6% triples consistent with the DBpedia-2016 [1] schema and 57.8% triples satisfy the property domain and range constraints. If a description logic reasoner is used to infer triples that logically follow an existing knowledge graph, and the graph contains inconsistent subset, it would stop the reasoner from accurately deducing logical consequences or drawing reliable inferences. The main aim of this thesis is to address the fundamental problems of "completeness" and "consistency" in the field of Knowledge Graph Completion by exploring a combination of conventional Knowledge Graph techniques for completion and semantic reasoning in an iterative way. Based on this, we explore the construction and enrichment of a domain-specific knowledge graph in a real-world scenario.

1. https://www.dbpedia.org/2016-10/

As expressed in the title of this thesis, “Schema Aware Knowledge Graph Completions”, there are two important notions that are investigated: (1) schema-aware, referring to the usage of a schema of a Knowledge Graph in determining the consistency of triples; (2) Knowledge Graph Completion (KGC), referring to our method of combining different Knowledge Graph Completion approaches with a semantic reasoning service, which are embedding-based KGC, a literal-embedding-based KGC, rule learning, and a materialisation service from an existing reasoner.

This thesis addresses three research questions:

Q1: How can we increase the number of consistent triples with respect to a schema of a Knowledge Graph that are produced by a Knowledge Graph Completion approach?
In addressing the first research question, we build a schema-aware KGC system, namely SICKLE. We employ various strategies at different levels. At the system level, we combined different types of triple producers and employ an approximated consistency checking service to produce schema-correct triples. Only schema-correct triples are added to the target KG and used in iterative training. We flexibly assemble KGC pipelines with four types of triple producers which learn new triples based on existing KG and a semantic reasoning service in an iterative manner and run in parallel or series mode. The different type of methods operate independently and are able to focus on their strengths and can benefit each other with new schema-correct triples fed back into next learning procedure. At the functional level, we integrate a literal-embedding-based methods. We found that knowledge from pre-trained language models enhances schema-awareness in KGC tasks, in particular, benefit schema-related KGC tasks, such as type prediction. At the algorithm level, we implement a schema-aware sampling strategy for the embedding-based methods. Schema-correct triples are sampled as positive examples and schema-inconsistent triples are sampled as negative examples. We observed that this sampling method not only have positive effect on producing schema-correct triples, but also improves performance in a downstream type prediction task. Unlike the existing approaches, that only focus on the completeness issue of a Knowledge Graph, our combination of approaches successfully produces more new triples that are consistent with regards to the schema of a Knowledge Graph.

Q2: How can we encourage models to produce more consistent triplets while maintaining accuracy in link prediction?
To tackle the second question, we optimize the combined KGC pipeline that address the first research question from two different directions. Firstly, we experiment with multiple data fusion methods, calibrating scores from different models and aggregating them into a final confidence score. We then use this fused score to re-rank the prediction results, leading to superior performance in the same link prediction task. Secondly, we extend the schema-aware negative sampling to open-world informative negative sampling. In essence, we utilize

schema to identify consistent and inconsistent subsets, and leverage the currently popular large language models to rank those negative samples within the consistent set that appeared more likely to be incorrect. Our negative sampling strategy outperforms two closed-world assumption (CWA) based sampling strategies in terms of link prediction results.

Q3: In a real scenario, given multiple complex data sources, how to make the a domain specific KG an evolving process by combining data-driven methods and knowledge-based methods, with regard to the schema of KG?

We explore an approach to obtain and complete knowledge in a real scenario within a 5G network log system. This involves extracting and predicting knowledge from a segment of 5G network system logs and making the evolution of the knowledge graph an iterative process. We break down this application work into two sub-tasks: construction and completion. The construction process can be regarded as a triple producer based on log extraction. We combine language models and Horn rules from background knowledge to learn triples from arbitrary logs, facilitating knowledge representation and reasoning. We implement a local-to-global strategy for triple inference, which reduces the reasoning query space. Then, with the constructed log KG, the completion process is where the SICKLE system is applied. We employ negative Horn rules from background knowledge and KG schema to assure the quality of learned triples. With the log extraction systems, we build a log KG and demonstrate its capability in root cause analysis. With the SICKLE system, we enrich the log KG with more schema-consistency triples.

Lay Summary

A Knowledge Graph (KG) is like a giant digital encyclopedia that connects information in a meaningful way, making it easier for computers to understand and process. Imagine it as a web of interconnected pieces of information, where each piece is represented as a "triple" consisting of three parts: a subject (aka head entity), a relationship, and an object (aka tail entity). For instance, in the triple "Apple is_a Fruit," "Apple" is the subject, "is_a" is the relationship, and "Fruit" is the object.

In a KG, schema and consistency are like the rules and order that help keep everything organized and reliable. Imagine the schema as a blueprint or guideline that outlines the structure and types of information allowed in the KG. It defines the types of entities (like people, places, or things) that can exist and how they relate to each other. For example, it might specify that a "Person" entity can have attributes like "Name" and "Age," and can be related to other entities like "WorksAt" or "LivesIn." Consistency, on the other hand, ensures that the information in the KG follows these rules and remains reliable. It means that the data is accurate, complete, and free from contradictions. For instance, if the KG says that "John works at Company X," it shouldn't later say that "John works at Company Y" unless there's a valid reason for the change.

Knowledge Graph Completion (KGC) is a method aimed at enhancing knowledge graphs by filling in missing entities or relationships and uncovering hidden facts within relational triples. While the success of knowledge graphs has led to increased interest in KGC methods, there remains a challenge in understanding the quality of generated triples, particularly in terms of schema consistency. Many existing KGC methods fail to ensure that the expanded knowledge graphs remain consistent with their original schemas, leading to difficulties in maintaining consistency while enhancing completeness.

This thesis, titled "Schema Aware Knowledge Graph Completions," explores the combination of conventional techniques for knowledge graph completion with semantic reasoning to address these challenges. The findings reveal insights into optimizing KGC pipelines for enhanced consistency and accuracy in link prediction. Furthermore, the study demonstrates the application of KGC methods in real scenarios, such as 5G network log analysis, showcasing the potential for enriching knowledge graphs with schema-consistent triples.

Acknowledgements

I would love to thank to my supervisors: Jeff Z. Pan, Alan Bundy and Wei Pang. They continuously guided, supported, and advised me during the years of my research. Without their relentless patience, sacrifice, and encouragement, I would not have been able to finish this thesis.

I would like to thank my parents, Changcun Wang and Guizhen Fang, their support means a lot to me.

I am grateful to my family, Jun Cheng, Christina Cheng, and Leo Cheng. With their support and encouragement, I have more faith in completing this work.

I am grateful to Xue Li, Ruiqi Zhu, Kwabena Nuamah, Lei Xu, Stefano Mauceri, Pasquale Minervini, Hiba Arnaout for all their knowledge, support and care throughout my journey.

I am very grateful to the panellist of my annual review: Mark Steedman, who have provided insightful comments on my work.

My endless thanks goes to Huawei for supporting the research on which this research was based under grant CIENG4721/LSC.

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Declaration

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

Fangrong Wang

Contents

Figures and Tables

Figures

Tables

Acronyms

ABox	Assertional part of a Knowledge Graph
ACC	Approximate Consistency Checking
CWA	Closed-world Assumption
DLs	Description Logics
IJPs	Inconsistent Justification Patterns
KG	Knowledge Graph
KGC	Knowledge Graph Completion
KGE	Knowledge Graph Embedding
LLM	Large Language Model
LP	Link Prediciton
MRR	Mean Reciprocal Rank
OWA	Open world Assumption
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
TBox	Terminological part of a Knowledge Graph
TP	Type Prediction
TREAT	Tacit Relation Extraction and Transformation dataset

Chapter 1

Introduction

1.1 Motivation

1.1.1 Complete a Knowledge Graph

In general, a Knowledge Graph can be seen as an ontology with an entity-centric view, consisting of a set of interconnected, typed entities and their attributes, as well as some schema axioms for defining the vocabulary (terminology) used in the Knowledge Graph [4]. The use of knowledge graphs [5, 6] has become popular in knowledge representation and knowledge management applications, including search [7–10], recommendation [11–17], medical informatics [18–21], finance [22–24], science [25–29], multi-modal content [30, 31], media [32–35], software engineering [36–41] and industrial domains [5, 42, 43].

Knowledge Graph Completion (KGC) aims to complete the structure of knowledge graphs by predicting the missing entities or relationships in them and mining unknown facts over the relational triples of the form $\langle h, r, t \rangle$, where h is the head entity, r the relation and t the tail entity. The KGC can be performed by either extracting new facts from external sources, such as textual documents, or by eliciting implicit facts within the existing content of the KG. The latter task is called *link prediction* (LP), the goal of which, given a knowledge graph, a head (tail) entity e and a relation r, is to predict the tail (head) entity $\langle h, r, ? \rangle$ ($\langle ?, r, t \rangle$). The general links that connect entities are usually inferred using techniques such as inductive learning, *Knowledge Graph embedding* (KGE) and rule mining. Under the LP category, there are at least two typical types of triple producers, such as Knowledge Graph embedding and rule learning [44–51]. Knowledge Graph embedding approaches, such as [52, 53] (and many others), complete an input Knowledge Graph ABox by learning vector representations of existing entities and relations for predicting missing relations. Rule learning approaches, such as [54, 55], complete an input Knowledge Graph ABox by learning rules based on patterns in the ABox. The learned rules can then be used to produce new relation assertions. The type links that add types of entities is called *type prediction* (TP). It is related to node classification task [56–58]. For example, training a classifier on the features of node and inward and outward edges to infer node types [59–61].

Correctness is another important factor during KG construction and completion. Correctness checking in KGs can be done by various methods, including probabilistic triple classification with machine learning or statistical methods [62–65], finding explanations from existing KG or external text evidence [66–69] and consistency checking with defined, external or mined constraints [70–73]. The triple classification methods and evidence mining methods usually do not take the schema of the KG into account, so that a triple that is classified as true might be inconsistent with the existing KG. For example, it could be the case that your contact address in school registration form (external text evidence) is different from that in your school contact (given $has_address$ is a functional property in an existing KG). Such inconsistency should be resolved by keeping the correct one.

There are many KGC approaches, each with its own strengths and limitations. The prominent approaches include: embedding-based methods, rule-based methods and materialisation that logically deduce triples from existing triples and schema. Although all of these KGC methods succeed in boosting the performance in tasks such as link prediction and triple classification, the existing KGC approaches share same challenges.

Knowledge Graph Embedding

Knowledge Graph embedding (KGE) is a technique to learn vector representations (embeddings) for entities and relations in such a way that the inherent structure and semantics of the knowledge graph are preserved. This enables the KGEs to perform tasks like link prediction, entity prediction, and relation prediction. For example, TransE [52] represents entities and relations as low-dimensional vectors in a continuous vector space. It assumes that the relationship between entities can be captured by the translation from the head entity to the tail entity.

To achieve higher performance in the link prediction and triple classification task, different KGE approaches propose different types of scoring functions (a function that is used for measuring how likely a triple holds in terms of the proximity between a head and a tail vector. The link prediction is done by replacing the head entity or tail entity with candidate entities, then calculating the scores of all the candidate entities and ranking the top k entities [45]. Whereas triple classification is a task that is used for discriminating between true and false triples. A summary of recent embedding-based models was reviewed and analyzed in Chapter 2.

The existing KGE approaches share some challenges that still cannot be addressed.

The first challenge is how to detect the inconsistencies of the knowledge that a Knowledge Graph captures. Because a Knowledge Graph is the result of a knowledge acquisition process that is prone to many errors, one can find many inconsistencies within a Knowledge Graph. The existing work relies on the process of embedding the entities and relationships into a vector space to eliminate these inconsistent facts. Of course this is not enough, therefore we

need other resources that can decide whether a piece of knowledge is consistent or not. A schema of a Knowledge Graph is an excellent resource to do this. It contains subsumption hierarchies of all concepts and properties that exist in a Knowledge Graph. It is also manually created and maintained by the domain experts.

Another challenge is the false negative problem involved in the KGE training procedure. Numerous strategies exist for generating negative samples in KGE. Classic uniform random sampling [52] randomly selects entities to replace either the head or tail of existing triple to create a negative sample. Other sampling strategies, such as Bernoulli sampling [74], importance sampling [75], adversarial negative sampling [76–78] and dual-encoder sampling [75] make efforts to generate more challenging negative samples that are difficult for the primary KGE model to distinguish. However, the majority of these negative sampling strategies are based on the *Closed World Assumption* (CWA), which presupposes that the absence of a fact (a triple) in the KG implies the falseness or nonexistence of that fact. This can lead to the false negative problem. For instance, if the KG contains the triple $(John, likes, IceCream)$, under the CWA, the negative sample $(John, likes, Pizza)$ would be unequivocally considered as false, suggesting that John does not like pizza. In contrast, the *Open World Assumption* (OWA) treats missing triples as unknown or unobserved.

The schema of a Knowledge Graph and its logical consistency is a way to alleviate the false negative problem: any triples not consistent with the existing KG are negative examples. A strand of research [65, 79, 80] focused on applying logic rules in their KGE sampling strategies, to address the false negative problem and also encourage KGEs to predict triples that adhere to certain constraints. While these negative sampling strategies have addressed the false negative problem to some degree, they either rely on limited constraints or sometimes include inconsistent triples and overlook statements that, although consistent, are incorrect.

How to incorporate consistency checking into a KGE approach is one of the core challenges addressed in this thesis. This incorporation can encourage a KGE approach in producing new and consistent triples.

Rule-based KGC

Another research direction of KGC is logic rule learning, which makes use of the symbolic nature of knowledge to identify or learn logic rules from an existing set of triples. Rules over KGs are of the form $head \leftarrow body$, where $head$ is an atom and $body$ is a conjunction of atoms. The rules are then applied to the existing KG to infer new triples.

Further, there has been an effort to combine the KGE approaches with the rule learning approaches to complete a Knowledge Graph. A prominent work in this area is [81]. They use the embedding model to evaluate the quality of the rules. The focus of the current rule learning approaches is to overcome the completeness issue of a Knowledge Graph by using the reasoning mechanism to infer new triples. This poses a challenge, how to increase the number of consistent triples that are produced by the rule learning approaches.

Literal-embedding-based KGC

Most of the KGE methods discussed in [45, 46, 82] are structure-based embeddings, which do not make use of any literal information of the entities. In fact, literals in a KG encode additional information which is not captured by the entities or relations. Literals can bring advantages to pure structure based KGEs by enriching the representation of entities and relations with semantic information. Also, literals are a natural substitute for missing topological features of novel entities or disconnected entities, hence they can help to learn embedding for novel entities [83].

Recent years have witnessed increasing interest in learning KG representations with extra literal information, such as entity names and descriptions [83–88]. The literal-embedding methods (L-method), such as DKRL [84], SSP [85], Conmask [83], BLP [87] and many others, have the potential of linking novel entities into an existing KG, which is referred as inductive LP [89]. In contrast, the conventional KGE approaches complete knowledge graphs with transductive LP [89], it means the full set of entities must be known during training.

Modern language models, such as BERT [90], automatically acquire knowledge from large-scale corpora via pre-training. Especially, phrases of the same type are relatively close in pre-trained text representation. For example, the cosine similarity between *Barack Obama* and *George W. Bush* from BERT encoding is 0.997, because both entities play similar roles and have similar context in text corpus. Previous literal-embedding-based KGEs focused on improving the LP accuracy based on the closed world assumption. There is no evaluation of whether the literal information would benefit the ontological schema-awareness. This poses a challenge: how to incorporate literal features, such as entity names and descriptions, and the knowledge stored in the pre-trained language models, to enhance the schema-awareness in KGC beyond the capabilities of standard KGE learning algorithms.

KGC Correctness

Because a KG is the result of a knowledge acquisition process that is prone to many errors, one can find many inconsistencies within a KG. The efforts to understand the quality of the candidate triples from the KGC methods, in particular from the schema aspect, have been limited. In fact, most existing KGC methods do not guarantee that the expanded KGs are consistent with the schema of the initial KG. Hence, schema-aware KGC seems to be the way to go.

Existing KGC research often uses the silver standard method [1] to measure the performance of Knowledge Graph completion approaches, which assumes that the KG itself is already of reasonable quality. In the silver standard method, some existing links in the data subgraph (ABox) are removed for testing if link prediction can help to recover the missing links. Our experiments bring the silver standard method into question. They show that only 89.9% triples are consistent with the original NELL-995 dataset [2] schema and satisfy the domain and range constraints. The corresponding ratios for the DBpedia Politics subset (DBped-P) [3] are 99.6% triples consistent with the DBpedia-2016 [1] schema and 57.8% triples satisfy the property domain and range constraints. The inconsistent triples contradict themselves or violate logical constraints. In other words, they contains information that cannot simultaneously hold true according to the defined schema or rules of inference. If we attempt to derive logical consequences from these datasets using traditional OWL reasoners [91], such as HermiT [91] and Konclude [92], the input KGs containing inconsistent data will result in an empty output set.

Also, previous research [93] shows that translation based methods, such as TransE, cannot properly capture simple rules; even Bilinear models, such as SimplE [94] and ComplEx [95], are severely limited when representing subsumption or equivalence between relations. This indicates that embedding based triple producers might have limited capability in terms of representing schema of Knowledge Graphs. If we apply embedding-based LP directly on a given KG, the percentage of the results that are consistent with the schema and satisfy the constraints is low.

How to incorporate inconsistency checking into a KGC pipeline and how to increase the number of consistent triples that are produced by a KGC pipeline are the core challenges addressed in this thesis.

1.1.2 Enriching a Real Scenario Log KG

Traditional methods of fault diagnosis through logs requires huge manual work from experts, which is not feasible for modern large-scale logs. Log analysis is a technique of deriving knowledge from log files [96]. It has been applied to a variety of applications such as anomaly detection [97–99], intrusion detection [100–103], and root cause analysis [104–108]. The diversity and cumulative nature of knowledge means that we cannot construct and complete KGs with just a single technique and in a single-pass. We envision an approach to obtain, organise and maintain knowledge automatically from a relatively small amount of data(compared with machine learning), so that we can build and enrich large-scale knowledge graphs. Leveraging the capability of both data-driven methods and knowledge based methods, we can have better intelligent systems than any systems built by purely one kind of method.

---

1. https://www.dbpedia.org/2016-10/

In a real enterprise 5G network scenario, the construction and enrichment of a knowledge graph involves several key steps. Initially, the process begins with extracting entities of interest from relevant data sources. These data sources can vary, ranging from structured databases to unstructured textual documents, and may encompass a wide range of domains and topics. Following entity recognition, relationships between entities need to be determined and represented in the graph. These relationships reflect the connections and associations between entities, allowing for a richer representation of knowledge. Although logs contain valuable information, deriving knowledge from logs is different from classic knowledge extraction against text: logs are weakly structured, vary in different formats and do not follow natural language grammar. Typical log analysis tasks start from template-based detection to parse useful information, such as entities and properties from logs [109]; then use data-mining algorithms to analyze or summarize patterns from events. However, these approaches do not represent events and their connections in formal knowledge representation nor do they consider other perspectives on data. Some recent studies have facilitated the integration of log information by linking knowledge extracted from logs and aligning them with a background knowledge graph (BKG) [101]. These approaches open up opportunities for downstream research and practice based on formally represented knowledge. For instance, it is possible to query the log KG and contextualize local event information. These approaches, however, only use the BKG as a target graph to align and merge new entities from logs, and do not leverage existing knowledge to infer implicit knowledge nor guarantee the quality of extracted knowledge.

How to learn knowledge from arbitrary logs and how to make the evolution of the knowledge graph an iterative process while considering the quality of learned triples is a challenge to be addressed in this thesis.

1.2 Research Questions

The main objective of our research is to explore how conventional Knowledge Graph Completion and semantic reasoning can be used together to iteratively enrich KGs with schema-consistent triples. To achieve it, we combine different types of KGC methods and produce schema-consistent triples as many as possible with respect to the schema of Knowledge Graph and to complete a given KG with only high ranked and schema-consistent triples. We also automate a process of building and enriching a 5G network knowledge model to facilitate knowledge representation and reasoning in an evolving manner. This includes construct a domain specific KG from arbitrary logs and complete it with schema-consistent triples.

Based on this objective, we aim to answer the following research questions:

- Q1: How to increase the proportion of consistent triples with regards to a schema of a Knowledge Graph that are produced by a Knowledge Graph Completion approach?

- Q2: How to encourage the KGC models to produce more consistent triplets while ensuring accuracy in link prediction?
- Q3: In a real 5G network scenario, given multiple complex data sources, how to make the a domain specific KG an evolving process by combining data-driven methods and knowledge-based methods, with regard to the schema of KG?

1.3 Thesis Outline

In this thesis, we go over our attempts to address the aforementioned research objectives and their corresponding research questions.

Chapter 2 presents all the related work that forms the basis of this research. This chapter includes an overview of Knowledge Graph, knowledge acquisition, Knowledge Graph reasoning, Knowledge Graph Completion. Also, we review related works in hybrid KGC systems and discussed how our work differs from others.

Chapter 3 introduces a system called SICKLE, which combines multiple KGC models and integrates an approximate consistency checking service to produce schema-correct triples. We address Q1 by employing a series of strategies to encourage the combined KGC pipeline to produce more schema-consistent triples. Additionally, we analyze and compare the effectiveness of these strategies in achieving the goal of generating schema-consistent triples.

Chapter 4 applies the SICKLE system on a real scenario domain-specific log KG. This application consists of two parts. First, we implement a log KG construction system called LEKG, which address Q3. LEKG extracts knowledge from a segment of real enterprise network log data. Then, we apply the SICKLE system to the extracted KG to generate more schema-consistent triples. LEKG combines various NLP approaches and rule inference techniques to extract, infer, and validate new triples from multiple sources.

In Chapter 5, we improve the SICKLE system through two separate research effort. Firstly, we delve into the combined methods within SICKLE by analyzing a range of KGC models and testing various combination strategies for the link prediction task. The purpose of this analysis is to gain a deeper understanding of the impact of different KGC methods and fusion strategies on the combined KGC pipeline. This work extends the SICKLE system by calculating a combined prediction confidence based on multiple KGC models, addressing Q2. Secondly, we introduce an open-world schema-aware negative sampling strategy for KGE models. This negative sampling strategy provides an alternative approach to addressing Q2. It builds upon SICKLE's schema-aware sampling strategy and addresses one of its limitations by considering both schema-inconsistent triples and schema-consistent negative triples. We leverage large language models to identify consistent but likely false triples and include them as negative samples in KGE training.

Chapter 6 summarizes the contributions of this thesis and outlines directions for future work.

1.4 List of Publications

Chapter 3 are based on the previously accepted work:

- Wang, F., Bundy, A., Li, X., Zhu, R., Nuamah, K., Xu, L., ... Pan, J. Z. (2023), Schema-aware Iterative Completion for Knowledge Graphs Revisited, Accepted by World Wide Web Journal, 2023

Chapter 4 is based on the previously published work:

- Wang, F., Bundy, A., Li, X., Zhu, R., Nuamah, K., Xu, L., ... Pan, J. Z. (2021). LEKG: A System for Constructing Knowledge Graphs from Log Extraction. ACM International Conference Proceeding Series, (i), 181–185. https://doi.org/10.1145/3502223.3502250

The content of Chapter 5 is planned to be published as an extension part of SICKLE plus.

Chapter 2

Background and Related Work

In this chapter, we review existing work on Knowledge Graphs, such as Knowledge Graph representation, Knowledge Graph reasoning, Knowledge Graph completion, description logics, and approaches to ontology debugging, all of which are important concepts in the context of Knowledge Graph. We also discuss the existing approaches for combining multiple KGC methods, especially the methods that combine KGE and schema features, and how our work differs from existing work.

2.1 Background Knowledge

2.1.1 Knowledge Graph

A Knowledge Graph is a structured representation of knowledge that captures information about entities (such as people, places, and things) and the relationships between them. Each entity might have various attributes. For example, the attributes of a person include name, birth date, nationality, etc. Entities are connected to each other by relations; e.g. you follow one of your colleagues in Twitter. Each entity needs an identification to distinguish one another. To facilitate the interlinking between various knowledge graphs, the entity IDs need to be globally unique. Types of entities and relations are defined in some machine-understandable dictionaries called ontologies. There are many ontology languages based on different knowledge representation formalism.

An ontology based on Description Logic (DL) comprises a TBox $\mathscr{T}$ and an ABox $\mathscr{A}$.

The ABox statements captured in the Knowledge Graph are of two forms:

- Relation assertion $r(h,t)$, where h is the head entity, r the relation and t the tail entity. In the triple format, it can be rewritten as $\langle h,r,t\rangle$.
- Type assertion $C(a)$, where a is an entity and C is a type. In the triple format, it can be rewritten as $\langle a, rdf:type, C\rangle$.

The TBox is a set of axioms such as General Concept Inclusion (GCI) axioms (e.g.,$C \sqsubseteq D$), Role Inclusion (RI) axioms (e.g., $r \sqsubseteq s$) and Inverse Role axioms (e.g., $s \equiv r-$), where C and D are concepts, r and s are roles, and $r-$ denotes the inverse of r. For example, the concepts of $River$ and $City$ being disjoint can be represented as $River \sqsubseteq \neg City$, or $River \sqcap City \sqsubseteq \bot$,where $\bot$ is the bottom concept representing an empty set.

In this thesis, we also consider a set of constraints $\mathscr{C}$ as part of the schema of Knowledge Graphs. The Shapes Constraint Language (SHACL) [1] is a World Wide Web Consortium (W3C) [2] language for the validation of Knowledge Graphs, which has been adopted by mainstream tools and triple stores. A SHACL schema is a collection of shapes, which are constraints that an ABox sub-graph should satisfy. Note that not all existing Knowledge Graphs have SHACL constraints. From a practical perspective, we reuse some axioms, such as domain and range, as constraints in the spirit of SHACL. The following example states that all values of $ex : has\_IRI$ need to be IRIs, at any subject.

Example 1. *A SHACL contraint:*
ex:NodeKindExampleShape
a sh:NodeShape ;
sh:targetObjectsOf ex:has_IRI ;
sh:nodeKind sh:IRI .

Throughout this thesis, we use the following definition of a Knowledge Graph:

A Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$, as consisting of a schema $(\mathscr{T} \cup \mathscr{C})$ and $\mathscr{A}$, which is a set of interconnected typed entities and their attributes (or ABox in description logic terminology). The schema of a Knowledge Graph consists of a set of ontological axioms $\mathscr{T}$ (or TBox in description logic terminology) that defines the vocabulary used in a KG, as well as a set of constraints $\mathscr{C}$.

The quality of a knowledge graph is crucial for its applications. For example, a knowledge graph should be consistent. For example, it could be the case that your contact address in your driving license is different than that in your school contact. To create a knowledge graph connecting these information, such inconsistency should be resolved by keeping the correct one. In addition to consistency, one also needs to consider correctness, and completeness (or coverage) of knowledge graphs, as well as efficiency and scalability of reasoning services for DL ontologies, such as subsumption testing and classification. Many of those aspects are related to the schema of a knowledge graph which consists of a set of logical formulas that express the axioms of a logical theory. It enables us to prove that these axioms are consistent or derive logical consequences.

---

1. https://www.w3.org/TR/shacl/
2. https://www.w3.org/

2.1.2 Knowledge Representation Standards for Knowledge Graphs

Description Logics

Description logics are a family of logic formalisms that describe the domain of discourse with so called concept and role expressions and their instances [110]. The fundamental modeling concept of a DL is the axiom, which is a logical statement relating roles and/or concepts. Description logics typically separate domain knowledge into two components, a terminological part called the TBox and an assertional part called the ABox. DLs are characterized by their expressive power, reasoning capabilities, and computational complexity. They offer a balance between expressivity and tractability, allowing for the representation of complex knowledge while providing efficient reasoning algorithms.

There are many varieties of DLs with different levels of expressiveness, such as $\mathcal{ALC}$, $\mathcal{ALUE}$, $\mathcal{SHI2}$, $\mathcal{SHIF}$, $\mathcal{SHOIN}$, or $\mathcal{SROI2}$ [3]. For reasoning purposes in a Knowledge Graph, the most expressive and widely accepted DL is $\mathcal{SROI2}$. Therefore, we chose this type of DL to conceptualize the notion of a Knowledge Graph throughout this thesis.

RDF and RDFS

RDF (Resource Description Framework) is a W3C standard originally designed as a data model for metadata. RDF provides a way to describe resources on the web and their relationships using subject-predicate-object triples. The subject, predicate, and object in RDF can be identified using Uniform Resource Identifiers (URIs), allowing for unique identification of resources and relationships.

RDF forms the basis for building Knowledge Graphs and expressing semantic relationships between entities and concepts. It enables the integration of data from multiple sources, making it easier to combine and query information across different domains. SPARQL (SPARQL Protocol and RDF Query Language) is the standard query language for RDF data. It allows for querying and retrieving specific information from RDF datasets using a pattern-based syntax.

RDFS provides a simple schema language for RDF, and allows one to declare classes/properties, using the predefined language level class $rdfs:Class/property$, $rdfs:Property$. In addition, RDFS can also specify some dependencies among classes and properties, using the predefined language level properties $rdfs:subClassOf$, $rdfs:subPropertyOf$, $rdfs:domain$ and $rdfs:range$:

It should be noted that RDFS only provides limited expressive power as a schema language. Its limitations includes:

- It does not support negation;

3. https://en.wikipedia.org/wiki/Description_logic

- It does not provide constructors to define classes;
- It does not support instance-level alignment; e.g. it cannot express that person:Obama is the same as politician:Barack_Obama.

But these limitations are addressed by the more comprehensive schema language OWL.

OWL

OWL (Web Ontology Language) is based on the principles of Description Logic, which is a family of formal logic systems used for representing and reasoning about knowledge. Syntactically, OWL can be regarded as an extension of RDFS with additional vocabulary predefined by the OWL schema [4]. For example, OWL adds numerous ontological constructs on top of those introduced by RDFS. Ontologies in OWL allow for the formal representation of concepts, relationships between concepts, and constraints on the relationships and attributes of entities. OWL enables the specification of complex class hierarchies, property restrictions, and logical axioms.

OWL has various profiles depending on different aspects of DL [111]. The latest version of OWL is OWL 2 [5], which has been recommended by the W3C as the de facto standard for Web ontologies. OWL 2 was the most famous ontology language. It aims to extend the expressiveness of the OWL specification by introducing new constructs. OWL 2 comes in different flavors or profiles, each offering a different level of expressiveness and computational complexity. The most commonly used profiles are OWL 2 EL (Existential Language), OWL 2 DL (Description Logic), OWL 2 QL (Query Language), OWL 2 RL (Rule Language) and OWL 2 Full. These profiles differ in terms of the constructs and reasoning capabilities they provide. Reasoning engines based on OWL ontologies can perform tasks such as class classification, instance checking, property inference, and consistency checking. These reasoning capabilities enable automated deduction and logical consistency verification in OWL-based systems.

OWL uses different terminologies compared to traditional DLs, in that it introduces concepts such as classes, object properties, and data properties, which are equivalent to the concepts of concepts, roles, and attributes, respectively, in Description Logics. In this thesis, we employ both terminologies based on contextual considerations.

---

4. http://www.w3.org/2002/07/owl
5. https://www.w3.org/TR/owl2-overview/

DL-Lite

DL-Lite [112] is a DL language specifically tailored to capture basic ontology languages, while keeping a low complexity of reasoning, in particular, reasoning is polynomial in the size of the instances in the Knowledge Graph. As usual in Description Logics, DL-Lite allows representing the domain of interest in terms of concepts, denoting sets of objects, and roles, denoting binary relations between objects. DL-Lite supports the following axioms:

1. class inclusion axioms: $B \sqsubseteq C$ where B denotes a basic concept $B ::= A \mid \exists R \mid \exists R^-$, C is a general class $C ::= B \mid \neg B \mid C_1 \sqcap C_2$, A denotes a named class and R denotes a named property;
2. functional property axioms: $Func(R)$, $Func(R^-)$, where R is a named property, and R^- is its reversed property;
3. individual axioms: $B(a)$,$R(a,b)$ where a and b are named individuals.

Compared to OWL 2, DL-Lite is simple from the language point of view, in which only membership of a concept or a role can be asked. DL-Lite is known for its excellent computational properties. It is designed to ensure tractability and efficient reasoning. DL-Lite supports efficient class classification, instance checking, and subsumption reasoning. However, it has limitations on representing complex relationships and axioms compared to the full expressiveness of OWL 2.

2.1.3 Knowledge Acquisition

Knowledge acquisition refers to the process of acquiring or extracting knowledge from various sources and representing it in a structured form that can be utilized by computer systems. It involves gathering information, understanding its meaning and context, and organizing it into a format that can be processed, stored, and accessed by Knowledge Graph systems [45, 113, 114].

Knowledge Extraction

Knowledge extraction refers to the process of automatically identifying and extracting structured information from unstructured or semi-structured sources, such as text documents, websites, databases, or other data sources [45, 113]. The goal is to transform the unstructured information into formally represented knowledge that can be easily processed and applied in downstream tasks such as causal analysis, textual entailment, question and answering, and more. There are several techniques and approaches used in knowledge extraction, such as:
Named Entity Recognition (NER): NER is the task of identifying and classifying named entities, such as people, organizations, locations, or dates, in text. It helps in identifying important entities for knowledge extraction.
Entity Linking: Entity linking aims to disambiguate and link named entities mentioned in text

to their corresponding entries in a knowledge base or a reference source. It connects the entity mentioned in the text with a unique identifier, enabling further enrichment and integration of knowledge.

Relation linking: This involves identifying and extracting relationships or connections between entities mentioned. For example, extracting relationships like "person A is married to person B" or "company A acquired company B."

Ontology and Knowledge Graph: Knowledge extraction often involves structuring the extracted information into ontologies or Knowledge Graphs. These representations provide a formal structure to organize and link the extracted knowledge, enabling efficient querying, reasoning, and analysis.

Log Analysis and Log Extraction

Typical log analysis tasks are designed to solve specific problems. Among these approaches, log representation in graphs has attracted recent research interest. Various graph-based approaches have been proposed in the literature, covering applications such as query log analysis [115], cybersecurity [116], anomaly detection [100, 117], root cause analysis [108], business process analysis [118]. These approaches focus on resolving problems by graph-theoretical methods, and do not aim to extract general knowledge from log data. Recent studies [101, 119] aimed at extracting general log data. These works involve not only a huge number of logs but also existing background knowledge. They rely on log parsing tools [120] to obtain event templates and then merge the lifted entities to a Backgroung Knowledge Graph by mapping and aligning based on similar vocabulary and common identifiers. Although these studies have a similar purpose to ours, they neither use the Background Knowledge Graph to infer implicit relations nor validate the extracted knowledge. [102] builds a system to integrate newly available structured data from public sources into a cybersecurity KG, which involves acquisition, extraction, lifting, linking, and validation steps. Its validation is to make sure the necessary properties lifted from logs are included for each generated individual. However, the approach only checks if a reference entity exist, rather than checking logical or semantic consistencies of the extracted knowledge.

2.1.4 Knowledge Graph Reasoning

Knowledge Graph reasoning refers to the process of inferring new knowledge or making implicit connections and inferences based on the existing information within a Knowledge Graph [49]. The goal of Knowledge Graph reasoning is to enhance the capabilities of Knowledge Graphs by uncovering hidden relationships, filling in missing information, resolving

inconsistencies, and making the Knowledge Graph more complete and coherent. It involves applying logical rules, inference techniques, and reasoning algorithms to draw conclusions and derive new knowledge from the structured data represented in the Knowledge Graph [121].

There are various types of reasoning techniques employed in Knowledge Graph reasoning. In the following subsections, we explain the main categories of Knowledge Graph reasoning methods related to this thesis.

Knowledge Reasoning based on First-order Logic

First-order logic reasoning involves applying logical inference rules to derive new knowledge from existing knowledge. For example, if it is known that "All humans are mortal" $(\forall x Human(x) \rightarrow Mortal(x))$, the inference rule allows instantiating it to "Socrates is mortal" by substituting Socrates for the variable x. [122] presents a query-time first-order reasoning approach for uncertain RDF knowledge bases with a combination of soft deduction rules and hard rules. Soft rules are used for deriving new facts, while hard rules are used to enforce consistency constraints among both Knowledge Graph and inferred facts.

Horn rules, named after logician Alfred Horn, are a specific form of logical rules widely used in knowledge representation and reasoning. Horn rules represent a subset of first-order logic rules that have special properties and can be efficiently processed. In the implication form, they have the following format: $A_1 \wedge A_2 \wedge ... \wedge A_n \rightarrow B$, where $A_1 \wedge A_2 \wedge ... \wedge A_n$ is the body of the rule and B is the head. A positive rule has an atom as its head, while a negative rule has the head $\perp$. Thus, a negative rule is in the form of $A_1 \wedge A_2 \wedge ... \wedge A_n \rightarrow \perp$. Positive Horn rules can help to generate new triples, and negative rules can help to identify contradicting triples [72, 123]. Horn rules can be manually scripted or mined from a given KG. By applying these rules to the KGs, new facts can be derived for complementing Knowledge Graphs and detecting errors. [124] proposed the AMIE system for mining Horn rules on a Knowledge Graph. It computes a set of non-monotonic rules, which subsequently can be revised by adding negated atoms to their bodies in order to account for exceptions. Traditional first-order inductive learner achieve high inference accuracy on small-scale KG, however, it is difficult to exhaust all inference patterns due to the complexity and diversity of entities and relations in large-scale Knowledge Graphs [46]. To solve this problem, a few works [72, 123, 125–127] use a series of pruning and query rewriting techniques for mining even larger Knowledge Graphs.

Knowledge Reasoning based on Ontologies

In this category, a reasoning process is very related to the ontology languages such as RDF/RDFS and OWL.

The complex entity or relational reasoning in a Knowledge Graph can be transformed into a consistency detection problem through TBox and ABox in DL, thus refining and realizing knowledge reasoning [128]. OWL 2, which is based on DLs, is a key standard schema language. OWL 2 provides rich expressive power, including a strong support for datatypes [129] and rules [130]. OWL 2 schema can be used to define class hierarchies, complex classes and relations, domain and range for relations, and more complex schema axioms. Logic languages allow deductive reasoning, such as consistency checking, materialization, query answering, as well as inductive and abductive reasoning.

Traditionally in expressive DLs, reasoning is performed by tableau-based algorithms [131]. While tableau algorithms provide a sound and complete approach to reasoning, they can be computationally expensive and exhibit intractability in certain cases. Tractable DLs such as the DL-Lite family [132] and EL family [133] enjoy a much lower complexity than expressive DLs [129].

In this thesis, several typical reasoning services performed with ontologies are utilized. These reasoning services include:

- Classification: Ontologies can be used to classify entities into different categories or classes based on their properties and relationships. This allows for efficient categorization and organization of knowledge. This service is used in Chapter 3 in the algorithm converting OWL 2 to DL-Lite. In essence, it determines subsumption and the class membership of individuals.
- Inference: Ontologies enable logical inference, where new knowledge can be inferred based on existing knowledge. This includes deductive reasoning, where conclusions are drawn from explicit statements, as well as inductive reasoning, where generalizations are made based on observed patterns. This service is used in Chapter 3 in the algorithm converting OWL 2 to DL-Lite. And in Chapter 4, it is used for rule-based relation linking, where Horn rules are used to derive implicit axioms.
- Query Expansion: Ontologies can expand queries by incorporating related concepts and relationships. This helps to retrieve more comprehensive and relevant information during information retrieval or search processes. This service is used in Chapter 3 for querying relational assertions with Konclude reasoner [92] and in Chapter 4 for querying entity types with Pellet reasoner [134].
- Consistency Checking: Ontologies can be used to check the consistency of knowledge by identifying contradictions or conflicts within the ontology. Inconsistent statements or relationships can be flagged, allowing for error detection and resolution. This service is used in Chapter 3, Chapter 4 and Chapter 5 for consistency checking.

- Materialisation: The process of deriving and generating the explicit facts or triples in a Knowledge Graph based on the information contained in an ontology. It involves populating the Knowledge Graph with inferred or implied knowledge that is not explicitly stated in the original data sources but can be logically deduced from the ontology's axioms, rules, and constraints. This service is used in Chapter 3.This service is used in Chapter 3 as a materialisation triple producer. Materialisation is different from classification service in that materialisation involves inferring a broader range of implicit knowledge, including class memberships, property relationships, and logical consequences, while classification focuses on organizing classes and determining subclass relationships.

In this thesis, we used a few DL reasoners:

- Pellet: Pellet is a sound and complete OWL-DL reasoner with extensive support for reasoning with individuals (including nominal support and conjunctive query), user-defined datatypes, and debugging support for ontologies [134]. We used it in Chapter 4 for rule-based relation linking, because Pellet also supports the inference of data property values which is not supported in HermiT and TrOWL.
- HermiT: HermiT is an efficient and highly optimized reasoner for OWL 2[91], based on optimised tableau algorithms. We compared our approximated consistency checking (ACC) service with HermiT in Chapter 3.
- TrOWL [135]: TrOWL supports for all the expressive power of OWL2-DL, while maintaining tractability, by using language transformations. As an approximated reasoner, TrOWL is faster than Pellet and HerMiT in all types of reasoning. We use TrOWL in Chapter 4 for materialisation and classification.
- Konclude [92]: Konclude is a high-performance reasoner for OWL2-DL. It implements a highly optimised version of the sound and complete tableau calculus enhanced with sophisticated preprocessing methods and tableau saturation. We use Konclude in Chapter 4 for materialisation and classification. Konclude uses its own API, which is different from OWLAPI inherited by Pellet, Hermit and TrOWL for creating, manipulating and serialising OWL Ontologies. We use Konclude reasoner for its fast reasoning speed, but need extra effort to convert ontology formats and query formats.

Knowledge Reasoning based on Embedding

Knowledge reasoning based on embedding refers to a paradigm in which knowledge representation and reasoning tasks are performed using embedded models. In this approach, knowledge is represented as a set of numerical vectors in a vector space. Each entity and relation in the Knowledge Graph is associated with a vector representation, often learned through techniques such as neural networks, tensor factorization, or word embeddings [48, 136].

The majority of existing KG embedding techniques focus on performing the embedding task using only observed facts. In these techniques, a Knowledge Graph is initially represented by randomly mapping entities and relations to vectors in a continuous vector space. A scoring function is defined for each fact in order to assess its plausibility. By maximizing the overall plausibility of observed facts, entity and relation embeddings can be obtained. We have listed the KGE models that are utilized in this thesis in Table 2.1.

Table 2.1: KGE models that used in this Thesis.

Model	Scoring function	Ent.& Rel. embedding
TransE [52]	$\|h+r-t\|_{\ell_{1/2}}$	$h,t,r \in \mathbb{R}^d$
RotatE [53]	$\|h \circ r - t\|$	$h,t,r \in \mathbb{C}^d$, $\|r_i\| = 1$
ComplEx [95]	$Re\left(h^T diag(r)\bar{t}\right)$	$h,t,r \in \mathbb{C}^d$
SimplE [94]	$\frac{1}{2}\left(h \circ rt + t \circ r't\right)$	$h,t,r,r' \in \mathbb{R}^d$
TuckER [137]	$\mathscr{W} \times_1 h \times_2 r \times_3 t$	$h,t \in \mathbb{R}^d_e$, $r,r' \in \mathbb{R}^d$
CP [138]	$\sum_{\mathbf{r}=1}^{\mathbf{R}} u^1_{\mathbf{r},h}.u^2_{\mathbf{r},r}.u^3_{\mathbf{r},t}$	$h,t,r \in \mathbb{C}^d$

Different KGE models capture different features and represent relations and entities by their unique structures and predict triples by various scoring functions. Each of these models has its own merits and demerit. The basic idea of TransE [52] is to represent the relations in a Knowledge Graph by considering the translations between entities in a continuous vector space. It assumes that if a triple is true, then the embedding of the head entity plus the embedding of the relation should be close to the embedding of the tail entity. Despite its simplicity and efficiency, TransE has limitations in dealing with 1-to-N, N-to-1, and N-to-N relations [74]. RotatE [53] is another translation-based model. RotatE defines each relation as a rotation from the head entity to the tail entity in complex space as $t = h \circ r$ where $\circ$ denotes the element-wise Hadamard product. RotatE can naturally model both symmetric and asymmetric relations. It represents the direction of a relation by encoding the rotation angle in the relation embedding, allowing it to capture both 1-to-N, N-to-1 relationships [45]. In RotatE, composite relations are assumed to be commutative because $f \circ r1 \circ r2 = f \circ r2 \circ r1$ in a complex space. But the relationship in reality is not necessarily commutative. ComplEx [95], CP [138], TuckER [137], and SimplE [94] belong to the Bi-linear models family. ComplEx represents entities and relations using complex-valued embeddings. It models relations as Bi-linear forms between the embeddings of the head and tail entities. Because of only using the real component, the model can fit asymmetric relations. However, the product of diagonalized relation matrices are commutative, so that the model cannot represent non-commutative relationships. SimplE [94] introduces the inverse of relations and calculates the average canonical Polyadia score of (h,r,t) and (t,r',h). TuckER introduces three-way Tucker tensor decomposition, and learns to embed by outputting a core tensor and embedding vectors of entities and relations. CP [138] represents a tensor as a sum of $\mathbf{R}$ rank one tensors.

Knowledge Graph Embedding with Literals

Most of the methods discussed in [45, 46, 82], including the state-of-the-art models such as RotatE [53], are structure-based embeddings, which do not make use of any literal information of the entities. In fact, literals in a Knowledge Graph encode additional information which is not captured by the entities or relations. Literals can bring advantages to pure structure based KGC by enriching the representation of entities and relations with semantic information. Literals are a natural substitute for missing topological features of novel entities or disconnected entities, hence they can help to learn embedding for novel entities [83]. There are different types of literals [88]:

- Text literals: a variety of information that sorted in knowledge bases in free text such as entity and relation names, descriptions, abstractions, comments, etc.
- Numeric literals: the information in the form of numeric such as date, geo-location, size, units and so on.
- Image literals: visual analysis can read useful information from images that are associated with entities.
- Other types of literals: useful external information linked to entities such as Wikipedia redirected URLs.

In this thesis, we focus on the models of KGE with text literals. There are a huge list of Knowledge Graph embedding models utilizing literals, the main differences between them are how to handle literals separately and how to incorporate literal features into entity embedding [88]. The literal embedding-based models, such as DKRL [84], SSP [85], learn and align structure-based and literal-based embeddings in a joint model. The ConMask [83] and MIA [86] are more of a triple extraction task for unseen entities. They learn embeddings of the entity's name and parts of its text-description to connect unseen entities to the KG. BLP [87] encodes an entity by mapping its description to a vector that acts as the embedding of the entity. The basic KGC learning algorithm is then carried out as usual. The BLP can be regarded as using a pre-trained model fine-tuned with link prediction. The entity representations learned via link prediction are also transferable to the tasks of node classification and information retrieval, which demonstrates that the entity embeddings act as compressed representations of the most salient features of an entity.

2.1.5 Knowledge Graph Completion

The following are the three tasks that are considered as the quintessential tasks of a Knowledge Graph Completion approach:

1. Link prediction aims to predict the likelihood or existence of a relationship between two entities that are not directly linked in the Knowledge Graph. Evaluation of link prediction models is typically done using rank-based metrics such as Hits@N and Mean Reciprocal Rank [1], or precision and recall [1]. These metrics assess the model's ability to accurately predict missing links between candidate entities.
2. Triple Classification in Knowledge Graphs refers to the task of assigning predefined categories or labels to individual triples in a Knowledge Graph. Evaluation of triple classification models is typically performed using metrics such as accuracy, precision, recall, or F1 score, which assess the model's performance in correctly assigning labels to triples [139, 140].
3. Type prediction in Knowledge Graphs refers to the process of automatically determining the most appropriate types or categories for entities in a Knowledge Graph. It involves analyzing the available information and relationships in the graph to infer the likely types or classes that an entity belongs to [56–58].

Metrics for Knowledge Graph completion evaluate the performance of KGC models in predicting missing or unseen facts in a Knowledge Graph. These metrics measure the ability of models to accurately capture the underlying relationships and generate plausible triples.

Here are some commonly used metrics for Knowledge Graph completion [1]:

1. Hits@N (also known as Hits@K): This metric measures the ability of a model to rank true triples higher than incorrect or false triples. It calculates the percentage of test triples where the true triple appears in the top-N ranked triples, where N represents the rank position. Common variations include Hits@1, Hits@3, and Hits@10. The higher the Hits@N scores, the better the performance of the link prediction task.
2. Mean Rank: This metric calculates the average rank of the true triple among all possible triples. A lower mean rank indicates better performance.
3. Mean reciprocal rank (MRR): MRR is also used for evaluating link prediction task, which calculates the average reciprocal rank of the true triple. A higher mean rank indicates better performance.
4. Precision, Recall and F1: Precision, recall, and F1 score are commonly used evaluation metrics for triple classification in Knowledge Graph completion tasks. These metrics assess the accuracy, completeness, and overall performance of models in classifying triples as positive or negative.
5. Accuracy: Accuracy measures the overall correctness of the classification and is calculated as the ratio of correctly classified triples to the total number of triples.

These metrics provide quantitative measures to assess the performance of Knowledge Graph completion models. Depending on the specific task and research objectives, different metrics may be more appropriate for evaluating the effectiveness and accuracy of the models. In this thesis, we used Hits@N, MRR, F1 in Chapter 3, Chapter 4 and Chapter 5.

2.2 Hybrid Knowledge Graph Completion Systems

2.2.1 Knowledge Graph Completion System Combined Multiple Models

Some recent work tries to combine KGE and rule-learning in a collaborative or complementary way. For instance, IterE [81] was designed to address the sparsity issue in embedding learning and the efficiency challenge in rule learning. It runs in an iterative manner in which rules are learned from embeddings and embeddings are learned from existing triples and new triples inferred by rules. A more recent study [141] constructs a simple method to combine the outcomes of a rule-based and a KGE approach in a post-processing step. This combination strategy ensures that the rule-based method and the KGE model operate independently, but interact by aggregating rankings, for example, using the KGE prediction scores as additional information to change the position in the ranking of a rule-based KGC.

None of these methods guarantee that an expanded Knowledge Graph is consistent with the ontological schema of the original Knowledge Graph.

2.2.2 Schema Enhanced KGE models

According to the analysis in [93], while bilinear models are fully expressive under specific embedding dimensionality bounds, they are severely limited in how they can only model sets of rules of the form $R(X,Y) \rightarrow S(X,Y)$. It, in particular, also means that they are not able to learn the type of dependencies captured by such rules.

A few studies [79, 142–144] aimed at learning joint models that inject logic into Knowledge Graph embeddings to obtain more predictive entity and relation embeddings. Approaches belonging to this category tightly integrate rule learning and embedding based approaches, but are mostly restricted to a type of rule which does not allow for constants. EmbedS [145], TransC [64], Cose [146] and OntoZSL [147] enrich their embedding models by considering ontological information, such as classes and hierarchy. Compared to these works, our system deals with a wider range of ontological schema axioms, such as symmetric relations, asymmetric relations, irreflexive, inverse of, domain, range, disjointedness and hierarchy. KALE [142] learns KGEs by jointly modelling translation based embedding and logic, where logical rules are represented as first-order logic formulae and modelled by t-norm fuzzy logics. TRANSOWL [65] and its variants, inject background knowledge during learning process by defining specific constraints, such as inverse of, equivalence, subsumption, on the energy functions for considered axioms. However, according to the analysis in [93], translation-based methods cannot properly capture simple rules. Some joint models may be limited by the expressive capability of their base translation-based models.

2.2.3 Schema-aware Knowledge Graph Completion

The previous work SIC [3, 70] regards the triples of a Knowledge Graph as being either schema-correct, schema-incorrect, or schema-unknown in terms of their compliance with the schema of that Knowledge Graph.

- **Schema-correct triples**: consistent with the schema of the Knowledge Graph and satisfying the constraints, such as domain and range;
- **Schema-incorrect triples**: are either not consistent with the TBox or not satisfying the constraints;
- **Schema-unknown triples**: they are consistent with the schema, but not yet satisfying the constraints, due to lack of some type information for their hs or ts, i.e., neither schema-correct nor schema-incorrect.

It considers both ontological TBox and SHACL constraints as parts of the schema of Knowledge Graphs. Accordingly, it considers both ontological consistency checking and SHACL constraints when define the notion of correctness.

SIC [3] exploys an iterative approach that leverages existing KGC methods and schema based logical reasoning for both producing triples and checking schema consistency. We revisited the notion of schema-aware KGC. While our core objective is similar to SIC, i.e., to produce schema-correct triples as many as possible with respect to the schema of Knowledge Graph and to complete a given KG with only schema-correct triples, we consider different strategies to achieve this objective.

Firstly, we develop a schema-aware sampling strategy to target the false negative problem. The closed-world assumption (CWA) assumes that triples present in the KG are true and triples not present in the KG are false. Under the CWA, a common silver standard sampling strategy uses the existing triples in a KG as positive examples, and creates negative examples by corrupting positive triples and replacing head or tail with randomly selected entities. However, the real-world KGs are usually dynamic and developing rapidly, it is hard to assume a given KG is perfect. Under the open-world assumption, the triples absent from the KG could be false negative. SIC tackled the false positive problem by only using schema-correct triples as positive examples, but it ignored the false negative problem. We tackle the false negative problem by developing a schema-aware negative strategy which generates schema-inconsistent triples and uses them as negative samples during KG embedding training procedure. By exploiting logical consistency in the sampling strategy, we would like to make high-rank triples as consistent as possible.

Secondly, we include a literal-embedding based KGC method, referred to as the L-method, in our combined pipeline. SIC completes knowledge graphs with transductive LP, where the full set of entities must be known during training. However, most real-world KGs evolve quickly with new entities and new triples being discovered over time. Recent years have witnessed

increasing interest in learning KG representations with extra literal information [83–88]. The literal-embedding methods (L-method), such as DKRL [84], SSP [85], Conmask [83], BLP [87] and many others, have the potential of linking novel entities into existing KG, which referred as inductive LP [89]. Our goal in integrating the L-methods is not just to use it to handle new nodes, but we argue that combining literals and the per-trained language models can improve schema-correctness in KGC pipeline compared to pure E-method.

Thirdly, we flexibly assemble KGC pipelines with four types of triple producers (an E-method, a R-method, an M-method and a L-method) and an Approximated Consistency Checking module in an iterative manner and run in parallel or series mode. The different type of methods operate independently and are able to focus on their strengths and can benefit each other with new schema-correct triples fed back into next learning procedure.

Furthermore, we designed the Approximated Consistency Checking module by transforming the schema in OWL 2 to DL-Lite to make the ACC more efficient. This allows us to integrate it into KGE's learning procedure for schema-aware sampling.

2.2.4 Knowledge Graph Ontology Embedding

The objective of Knowledge Graph ontology embedding is to encode the semantics of an ontology by taking into account its graph structure, lexical information and logical constructors. It differs from Knowledge Graph Embedding as it focuses more on the TBox [61]. KGE, on the other hand, typically focuses on the ABox.

[148] proposed modelling the semantics of the logic constructor using geometric learning from the description logics $\mathscr{EL}^{++}$ [149]. It embeds classes as n-balls in n-dimensional space and relations as n-dimensional vectors and maps a class to the radius associated with it. The embeddings are generated through optimization using a set of loss functions corresponding to different normal forms of the axioms in ontologies; such normal forms can be generated for ontologies formalized in the Description Logic EL [133], but may not exist for other, more expressive logics [150].

Onto2Vec [151] and OPA2Vec [152] generates embeddings for ontology classes and instances, considering the logical axioms in named classes. Both of them adopt the deductive closure of an ontology with entailment reasoning, mainly subsumption between named classes and treat each axiom in the ontology as a sentence and learn a word embedding model which encodes a statistical correlations between items in a corpus. These approachs limits their ability to explore more complex correlations between axioms.

OWL2Vec [153] projects ontology to a graph and captures the semantics of OWL ontology by exploring the neighborhoods of classes, then generates embeddings using Word2Vec. Unlike Onto2Vec and OPA2Vec, which directly convert axioms to sentences, OWL2Vec creates a corpus of sentences according to walking strategies. As a result, it scales up the corpus

and captures the graph structure. $OWL2Vec^*$ [61] is an upgraded version of OWL2Vec. It incorporates an OWL entailment reasoner to materialise the ontology and approximates an OWL (or OWL 2) ontology to RDF triples, taking account logic constructors such as class disjointness, existential and universal quantification (e.g., a person only has one date of birth) [61]. It then walks over its graph forms and generates a corpus of documents that capture different aspects of the semantics of the ontology, including a structure document, a lexical document, and a combined document where ontology entailment reasoning can be enabled.

While Onto2Vec, OPA2Vec and $OWL2Vec^*$ have performed well in ontology relevant prediction tasks, such as class subsumption prediction and class clustering, their weakness lies in the non-contextual word embedding model they adopt which has been proven to be less effective than more recent Transformer-based contextual language models like BERT in many natural language understanding and sequence learning tasks [154]. This limitation has been addressed in BERTSubs [154]. BERTSubs models concept subsumption prediction as a downstream classification task for BERT fine-tuning, with transforming two target concepts and their contexts including the neighbouring concepts and existential restrictions into a pair of sentences as the input.

Our work shares similarities with these studies in that we incorporate graph structure, lexical information, and logical constructors into our KGC pipeline. However, these studies are primarily oriented towards ontology-tailored text embedding models and are proficient in predicting class subsumption within OWL ontologies, a goal distinct from that of our thesis.

2.2.5 Negative Sampling Strategy for KGE

Uniform random sampling [52] is the easiest and most widely used negative sampling strategy in KGE. It randomly selects positive entities to replace either the head or tail to create a negative sample. While uniform random sampling is easy to process, it may produce low-quality negative samples, leading to a noisy training signal. TransR [74] uses the Bernoulli distribution to determine whether to replace the head or tail entity in a positive triple with a random entity. It gives more chance of replacing the head in 1-to-many relations and the tail in many-to-1 relations. While this strategy is an improvement over uniform random sampling, it still suffers from the issue of generating overly simple, false negatives. Importance Sampling [75] and dual-encoder sampling [75] assign higher probabilities to entities that are more informative or relevant as negative samples, based on their frequency, importance in Knowledge Graph or relevancy to entities in a positive triple. Adversarial negative sampling [76–78] involves the use of a separate adversarial model to generate challenging negative samples that are hard for the primary KGE model to distinguish from positive data. All these negative sampling strategies are based on the CWA and cannot avoid the false negative problem.

A strand of research focused on applying logic rules in their KGE sampling strategies. [79] proposed a method to encode logical consistency into the distributed representation to make high-rank triples as consistent as possible. TRANSOWL [65] avoids false positives by exploiting available axioms specified in RDFS and OWL to generate negative examples. [80] proposed a method that leverages the schema to dynamically generate inconsistent triples as negative examples in its training procedure. These sampling strategies are similar to our schema-aware negative sampling strategy, in that ontological knowledge is taken into account for the generation of negative samples. But our work differs in considering those consistent, but likely to be false, examples and expands the negative sampling strategy to an open-world assumption.

The task of negative generation using language models [155, 156] or relying on peer-based statistical inferences [157–159] primarily falls within the domain of Natural Language Processing (NLP), with a focus on enhancing the informativeness of negative statements. In contrast, we focus on generating negatives specifically tailored for KGE training. Additionally, we utilize the Knowledge Graph schema to aid in the identification of negative triples derived from a baseline KGC system. This differs from the negative generation task, which typically does not make use of the schema or only considers a limited set of constraints.

2.3 Advantage over State of the Arts

In this section, we discuss the literature that is related to this thesis. The work presented in this thesis is most related to those studies addressing combining logical rules and KGE models for Knowledge Graph construction and completion. There are a few trends of assemble strategies:

1. [64, 65, 79, 142–147] aimed at learning joint models that inject logic into KGEs to obtain more predictive entity and relation embeddings. They only deal with a limited range of ontological information and do not use the schema to decide whether a new triple is consistent or not.

   We address this limitation by developing an approximate consistency checking algorithm to identify inconsistent subsets in the produced triples. This ensures that only schema-correct triples are included in an expanded KG. To efficiently identify inconsistent triples, we rely on the description logic expressions that can be found or inferred in the schema of a Knowledge Graph in OWL 2, which is a rich Description Logic with enhanced expressive power.
2. [81, 141] combine KGE and rule-learning in a collaborative or complementary manner. None of these methods guarantee that an expanded Knowledge Graph is consistent with the ontological schema of the original Knowledge Graph.

We have assembled KGC pipelines with various types of KGC models, including embedding-based KGC, rule-based KGC, literal-embedding-based KGC and materialisation. These KGC models are combined in a post-processing manner, where the newly generated schema-correct triples are merged into KG and fed into the next round of training.

3. SIC [3] combines multiple KGC models and uses the schema to exclude those inconsistent new triples. The identified erroneous data is discarded, but it could have served as valuable negative sampling examples to induct the KGC models produce more consistent triples.

   In our work, we make use of the schema-inconsistent triples identified in each iteration, so the schema-aware negative sampling strategy and KGE training become a convenient and natural combination. The aim of this combination is to increase the coverage (the number of triples) of a Knowledge Graph while maintaining its consistency.

4. [65, 79] have focused on applying logic rules in their KGE sampling strategies to avoid the false negative problem and force KGE model to produce more consistent triples. But it only uses a small set of inconsistent triples in the optimal objective function for consideration of efficiency and scalability. [80] proposed a method that leverages schema to dynamically generate inconsistent triples as negative examples in its training procedure. However, it requires training the model once in advance to generate adversarial negative samples, which can be cumbersome. Moreover, these sampling strategies ignore those yet consistent but negative triples.

   To tackle this limitation, we develop a schema-aware open-world sampling strategy by applying consistency checking and making use of large language models to identify and rank negative triples. By doing this, we avoid the false negative problems that are caused by CWA. We extend the closed-world sampling strategy to a schema-aware open-world sampling strategy.

Chapter 3

SICKLE: Schema-aware Iterative Completion for KnowLEdge Graphs

3.1 Background

3.1.1 Schema-aware KGC

Given a Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$, SICKLE uses schema $(\mathscr{T} \cup \mathscr{C})$ to supplement a set of schema-correct triples for the data sub-graph $\mathscr{A}$. The previous work [3, 70] regards the triples of a Knowledge Graph as being either schema-correct, schema-incorrect, or schema-unknown in terms of their compliance with the schema of that Knowledge Graph. We add two more categories, schema-consistent and schema-inconsistent to better describe the different levels of outputs from SICKLE's approcimate consistency checking (ACC) module.

- **Schema-correct triples**: consistent with the schema of the Knowledge Graph and satisfying the constraints, such as domain and range;
- **Schema-incorrect triples**: are either not consistent with the TBox or not satisfying the constraints;
- **Schema-unknown triples**: they are consistent with the schema, but not yet satisfying the constraints, due to lack of some type information for their hs or ts, i.e., neither schema-correct nor schema-incorrect;
- **Schema-consistent triples**: they are consistent with the schema, and not proved to violate the constraints, i.e, either schema-correct or schema-unknown;
- **Schema-inconsistent triples**: not consistent with the schema.

For example, given a set of schema axioms and type assertions:

- $Domain(at\_school) = Student$, $Range(at\_school) = School$, $School \sqcap Person \sqsubseteq \bot$
- $Student(jim)$, $Person(anna)$, $School(westhill\_primary)$

We have three triple examples:

1. $\langle Jim, at\_school, westhill\_primary \rangle$ is schema-correct;
2. $\langle Jim, at\_school, anna \rangle$ is schema-inconsistent;
3. $\langle anna, at\_school, westhill\_primary \rangle$ is schema-unknown, yet schema-consistent.

A schema-incorrect triple is not consistent with the schema of a Knowledge Graph, or does not satisfy the constraints. While a schema-consistent triple is either schema-correct or schema-unknown.

Definition 1. *Given a Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$, a triple (h,r,t), where h and t are entities and r is an object property in $\mathscr{G}$, with C_h, C_t being some types of h and t resp., and D_r, R_r being some domain and range of r. We say (h,r,t) is a schema-correct triple w.r.t. $\mathscr{G}$ if:*

1. *the TBox and expanded ABox $\big(\mathscr{T} \cup \mathscr{A} \cup (h,r,t)\big)$ is consistent, and*
2. *$C_h \equiv D_r$ and $C_t \equiv R_r$ (domain and range constraints in $\mathscr{C}$).*

In Definition 1, the domain and range are used as constraints in $\mathscr{C}$.

Definition 2. *Given a Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$, a triple (h,r,t), where h and t are entities and r is an object property in $\mathscr{G}$, with C_h, C_t being some types of h and t resp., and D_r, R_r being some domain and range of r. (h,r,t) is a schema-incorrect triple with regard to $\mathscr{G}$ if:*

1. *$\mathscr{T} \cup \mathscr{A} \models (h,r,t) \sqsubseteq \bot$, or*
2. *$\mathscr{T} \cup \mathscr{A} \models C_h \sqcap D_r \sqsubseteq \bot$, or*
3. *$\mathscr{T} \cup \mathscr{A} \models C_t \sqcap R_r \sqsubseteq \bot$*

Definition 3. *Given a Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$, a triple (h,r,t), where h and t are entities and r is an object property in $\mathscr{G}$, (h,r,t) is a schema-unknown triple with regard to $\mathscr{G}$ if it is neither schema-correct nor schema-incorrect with regard to $\mathscr{G}$.*

Definition 4. *Given a Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$, a triple (h,r,t), where h and t are entities and r is an object property in $\mathscr{G}$, (h,r,t) is a schema-consistent triple with regard to $\mathscr{G}$ if $\big(\mathscr{T} \cup \mathscr{A} \cup (h,r,t)\big)$ is consistent.*

Definition 5. *Given a Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$, a triple (h,r,t), where h and t are entities and r is an object property in $\mathscr{G}$, (h,r,t) is a schema-inconsistent triple with regard to $\mathscr{G}$ if $\mathscr{T} \cup \mathscr{A} \models (h,r,t) \sqsubseteq \bot$.*

Our core objective is to produce "correct" triples as many as possible with respect to the schema of a KG. SICKLE's schema-aware KGC pipeline expands a target KG with only schema-correct triples. It uses an ACC module to not only detect the inconsistencies but also identify schema-correct triples that either already exist in the Knowledge Graph, or are produced by triple producers in an iterative process, so that only new schema-correct triples will be added to the target KG. We also incorporate schema-awareness in the KGE training process by implementing the schema-aware sampling strategy, where only schema-correct triples are selected as positive examples and schema-inconsistent triples are preferred to be selected as negative examples. These definitions are used in our evaluation in Section 3.8.

3.1.2 Combined and Iterative KGC

Two of the most common KGC methods are KG embedding (E-method) and logic rule learning (R-method). As discussed in [45], typical embedding-based models first learn embedding vectors based on existing triples. The prediction is done by replacing the head entity or tail entity with candidate entities, then calculating the scores of all the candidate entities and ranking the top k entities. A summary of recent embedding based models were reviewed and analyzed in [44–46, 82, 160, 161]. The logic rule learning makes use of the symbolic nature of knowledge to identify or learn logic rules from an existing set of triples. Rules over KGs are of the form $head \leftarrow body$, where $head$ is an atom and $body$ is a conjunction of atoms. The rules are then applied to the existing KG to infer new triples.

Both the embedding-based method and the rule-based method attempt to capture patterns present in KGs and generalize them so as to infer new triples, but they have differences.

The underlying patterns captured by the E-methods are hidden in the models. The expressive capability of KG embeddings are related to the designed characteristics of KGE models [161, 162] and distributions of the training data [81]. [81] pointed out that one of the main challenges of embedding learning is encoding sparse entities, in that the prediction results of sparse entities are generally worse than those of frequent ones.

Unlike the KG embedding, the logic rule learning is explainable and can provide insights for inference results. Given a target relation r, a straightforward approach of rule learning is to look for triples $< a, r, b >$ and search possible paths up to a certain length between a and b. Using the paths as rule body, the confidence of candidate rules is calculated by dividing number of supports with number of groundings with relation r in the KG. When applying rules to infer new triples, given a completion task $< a, r, ? >$, the R-methods select rules with r in its $head$ and replace head entity with a, then search all body groundings in a given KG [162]. When learning logic rules and applying them on KGC, the R-methods require many extensive searches on a given KG. Although the R-methods usually have some mechanisms, such as limiting the length of paths, to balance running time and completeness, the huge search space is still a challenge [72, 126, 163].
Different from the E-methods and the R-methods, the M-method infers new triples from the given KG, where all axioms that logically follow $(\mathscr{T} \cup \mathscr{C}, \mathscr{A})$ in $\mathscr{G}$ are materialised as new axioms. In [110], the materialisation is defined as below:

Definition 6 (Materialisation)**.** *For an ontology $\mathscr{O}$, its ontology materialisation is the set $\{A \sqsubseteq B \mid A, B \in N_C, \mathscr{O} \models A \sqsubseteq B\} \cup \{a : A \mid A \in N_C, a \in N_I, \mathscr{O} \models a : A\} \cup \{(a,b) : r \mid r \in N_P, a, b \in N_I, \mathscr{O} \models (a,b) : r\}$, where N_C, N_P and N_I are named classes, named properties, and individuals in $\mathscr{O}$.*

In SICKLE, we also include a literal-embedding model, referred to as L-method. Following [87], the input KG of the L-methods is defined as a tuple $\mathscr{G} = (N_I, N_P, Tr, \mathscr{D})$, consisting of a set of individuals N_I, a set of named properties N_P, schema-correct triples S_{cor}, and entity descriptions $\mathscr{D}$. For each entity $e_i \in N_I$, there exists a description $d_{ei} = (w_1, ..., w_n) \in \mathscr{D}$, where all w_i are words in a vocabulary V. Given a Knowledge Graph $\mathscr{G} = (N_I, N_P, Tr, \mathscr{D})$, the L-methods are able to complete $\mathscr{G}$ by producing a set of missing triples $Tr' = \{r(h,t) \mid r(h,t) \notin Tr, h \in N_I', t \in N_I', r \in N_P\}$ in the incomplete Knowledge Graph $\mathscr{G}$ where N_I' is an entity superset. In terms of our research objectives, we investigate if the literal features, such as the entity names and descriptions, and the knowledge stored in the pre-trained language model would improve the schema-awareness in KGC, compared to basic KG embedding learning algorithms.

3.2 Problem Statement

This chapter is to address the first research question.
Q1: How to increase the proportion of consistent triples with regards to a schema of a Knowledge Graph that are produced by a Knowledge Graph Completion approach?

We address this research question by considering the following strategies and hypothesis:

- We developed a schema-aware sampling strategy, which targets the false negative problem in common sampling strategy of CWA. We generate schema-inconsistent triples as negative examples during KG embedding training procedure, so that false negatives can be avoided. In addition, we only use schema-correct triples as positive examples in training and only schema-correct predictions are added to a target KG. Our hypothesis is that the schema-aware sampling strategy would improve the KGC pipeline in producing schema-correct triples.
- We include a literal-embedding based model in the system. Our hypothesis of this strategy is that the entity literals and the knowledge stored in a pre-trained language model would improve the schema-awareness when producing new triples.
- We assemble KGC pipelines with different types of triple producers (an E-method, a R-method, an M-method and a L-method), because we believe that different methods can benefit and complement each other. The combined pipelines can be executed in an iterative manner, so that different types of triple producers can benefit each other with new schema-correct triples fed back to learning procedure. Our hypothesis is that the KGE's schema features can be enhanced by injecting new schema-correct data in combined and iterative training.

- We execute pipelines in both the series mode and the parallel mode. In different execution modes, pipelines share data in different ways. We would like to know how these two different execution settings would impact the performance of pipelines in producing schema-correct triples.

Our strategies are designed to enhance schema-aware features in KGC pipelines, so that the pipelines tend to produce more schema-correct triples. To evaluate the efficacy of these strategies, we designed two tasks. One is producing schema-correct triples via LP which is , another is TP which is implemented as a multiple classification task. We assemble KGC pipelines with these strategies and their combinations to produce relation assertions via LP task. We would like to know their efficacy in producing schema-correct triples. Our TP task is a downstream task of LP, which uses the KGEs learned in LP as input features and predicts entity types. In a combined and iterative pipeline, the KGEs are learned with more schema-correct triples in iterations. We would like to know whether this data enhancement has advantages in downstream TP task.

3.3 Approximate Consistency Checking

The conventional reasoning services for Description Logic ontologies, are usually expensive, due to the high reasoning complexity [110]. The consistency checking services provided by OWL reasoners, such as HermiT [131] and TrOWL [135], are relatively fast to detect if a given KG is consistent or not, however, the justification service that identifies explanations for inconsistencies in a KG is time consuming. In the iterative pipeline, the expanded Knowledge Graphs can become a lot larger than the original Knowledge Graph. Due to the scalability issues of sound and complete reasoners, we decided to consider the approximate consistency checking (ACC) method that was introduced in [70]. This approach learns a few inconsistent justification patterns (IJPs) from TBox, and then uses these patterns to justify inconsistent subsets in ABox. However, one drawback of ACC, as described in [70], is that it relies only on explicit constraints defined in the given KG, such as domain, range and class disjointedness, and ignores deeper reasoning. This drawback makes it difficult for it to be applied to rich expressive ontologies and hard to identify complex inconsistent explanations from the expanded KGs in an iterative pipeline. To tackle this problem, we designed a new ACC module to support a certain level of multi-hop reasoning. Our notion of consistency checking in this thesis relies on two criteria: (1) is able to identify explanations with a certain level of multi-hop reasoning; (2) can quickly process large KGs, so that it is possible to integrate the ACC into KGC pipelines.

Example 2. *Given these axioms:*
$Range(has\_parent) = Person$
$Range(works\_for) = Organisation$
$DisjointWith(Person, Organisation)$
$has\_parent(Anna, John\_Lewis)$
$works\_for(Mary, John\_Lewis)$

In Example 2, the designed IJPs, such as $(A(a), Range(r) = R, R \sqcap A \sqsubseteq \bot)$ in [70], is not able to identify
$\{has\_parent(Anna, John\_Lewis), works\_for(Mary, John\_Lewis)\}$ as a schema-inconsistent subset, because entity $John\_Lewis$ doesn't have type axioms in this example. However, based on the CWA, we can infer and obtain axioms in Example 3.

Example 3. *Inferred axioms:*
$\exists has\_parent^- \sqsubseteq Person$
$\exists works\_for^- \sqsubseteq Organisation$
$\exists has\_parent^- \sqcap \exists works\_for^- \sqsubseteq \bot$

Let's design a new IJP: $\exists r1^- \sqcap \exists r2^- \sqsubseteq \bot$. In Example 3, the disjointedness axiom $\exists has\_parent^- \sqcap \exists works\_for^- \sqsubseteq \bot$ matches the new IJP $\exists r1^- \sqcap \exists r2^- \sqsubseteq \bot$.
The new IJP is able to identify the schema-inconsistent subset in Example 2: any subset in the form of $(works\_for(x,e), has\_parent(y,e))$ is a schema-inconsistent subset. Then we can justify that
$\{has\_parent(Anna, John\_Lewis), works\_for(Mary, John\_Lewis)\}$ is a schema-inconsistent subset.

The basic idea of our ACC approach is that we calculate a series of IJPs from the TBox with such reasoning power; then we scan and match IJPs in an ABox to detect inconsistencies that either already exist in the KG, or are introduced by triple producers in an iterative process.

To calculate IJPs, such as $\exists has\_parent^- \sqcap \exists works\_for^- \sqsubseteq \bot$, we apply an idea of knowledge compilation [164, 165] by semantically approximating a source ontology $\mathscr{O}_1$ in a more expressive DL $\mathscr{L}_1$ (source language OWL 2) with its least upper-bound $\mathscr{O}_2$ in a less expressive DL $\mathscr{L}_2$ (target language DL-Lite). In our approach, we only apply the knowledge compilation on the TBox, namely *TBox transformation* [110].

DL-Lite [112] is a Description Logic language specifically tailored to capture basic ontology languages, while keeping a low complexity of reasoning, in particular, reasoning is polynomial in the size of the instances in the Knowledge Graph. As usual in Description Logics, DL-Lite allows representing the domain of interest in terms of concepts, denoting sets of objects, and roles, denoting binary relations between objects. DL-Lite supports the following axioms:

1. class inclusion axioms: $B \sqsubseteq C$ where B denotes a basic concept
   $B ::= A \mid \exists R \mid \exists R^-$, C is a general class $C ::= B \mid \neg B \mid C_1 \sqcap C_2$, A denotes a named class and R denotes a named property;

2. functional property axioms: $Func(R)$, $Func(R^-)$, where R is a named property;
3. individual axioms: $B(a)$,$R(a,b)$ where a and b are named individuals.

Compared to OWL 2, DL-Lite is simple from the language point of view, in which only membership of a concept or a role can be asked. After the TBox transformation, some OWL 2 property characteristics, such as
$Domain(r)$, $Range(r)$, $InverseOf(r1,r2)$, $SubperpertyOf(r1,r2)$, $DisjointWith(C1,C2)$, are normalized to class subsumption and role subsumption in DL-Lite syntax. Hence we can easily look for the TBox IJPs in the form of subsumption axioms, such as
$(\exists works\_for^- \sqsubseteq \neg\exists has\_parent^-, \exists has\_parent^- \sqsubseteq \neg\exists works\_for^-)$.

Then we scan the ABox axioms to identify the ABox IJPs, such as the subsets in the form of $(works\_for(x,e), has\_parent(y,e))$. The approximated TBox preserves rich information in the original TBox. We designed a list of IJPs with DL-Lite syntax in Table 3.1, so that we can easily calculate simple but relatively comprehensive IJPs from an approximated TBox in DL-Lite and detect inconsistent subsets from an ABox. The details of the algorithms will be further described in Section 3.7.3.

Table 3.1: Inconsistency Justification Patterns

ID	TBox subset of the Pattern	ABox subset of the Pattern
1	$\exists r \sqcap A \sqsubseteq \bot$	$\langle e1,r,e2\rangle, \langle e1,rdf:type,A\rangle$
2	$\exists r1 \sqcap \exists r2 \sqsubseteq \bot$	$\langle e1,r1,e2\rangle, \langle e1,r2,e3\rangle$
3	$\exists r1 \sqcap \exists r2^- \sqsubseteq \bot$	$\langle e1,r1,e2\rangle, \langle e3,r2,e1\rangle$
4	$\exists r1^- \sqcap A \sqsubseteq \bot$	$\langle e2,r1,e1\rangle, \langle e1,rdf:type,A\rangle$
5	$\exists r1^- \sqcap \exists r2 \sqsubseteq \bot$	$\langle e2,r1,e1\rangle, \langle e1,r2,e3\rangle$
6	$\exists r1^- \sqcap \exists r2^- \sqsubseteq \bot$	$\langle e2,r1,e1\rangle, \langle e3,r2,e1\rangle$
7	$FunctionalProperty(r)$	$\langle e1,r,e2\rangle, \langle e1,r,e3\rangle$
8	$FunctionalProperty(r1), r2 \sqsubseteq r1$	$\langle e1,r1,e2\rangle, \langle e1,r2,e3\rangle$
		$\langle e1,r2,e2\rangle, \langle e1,r2,e3\rangle$
9	$FunctionalProperty(r1), r2^- \sqsubseteq r1$	$\langle e1,r1,e2\rangle, \langle e3,r2,e1\rangle$
		$\langle e2,r2,e1\rangle, \langle e3,r2,e1\rangle$
10	$Asymmetric(r)$	$\langle e1,r,e2\rangle, \langle e2,r,e1\rangle$
11	$Asymmetric(r1), r2 \sqsubseteq r1$	$\langle e1,r1,e2\rangle, \langle e2,r2,e1\rangle$
		$\langle e1,r2,e2\rangle, \langle e2,r2,e1\rangle$
12	$Asymmetric(r1), r2^- \sqsubseteq r1$	$\langle e1,r1,e2\rangle, \langle e1,r2,e2\rangle$
		$\langle e2,r2,e1\rangle, \langle e1,r2,e2\rangle$
13	$Irreflexive(r)$	$\langle e,r,e\rangle$
14	$A_1 \sqcap A_2 \sqsubseteq \bot$	$\langle e1,rdf:type,A_1\rangle, \langle e1,rdf:type,A_2\rangle$
15	$A_1 \sqcap \exists R_x \sqsubseteq \bot$	$\langle e1,rdf:type,A_1\rangle, \langle e1,r_x,e2\rangle$
16	$A_1 \sqcap \exists R_x^- \sqsubseteq \bot$	$\langle e1,rdf:type,A_1\rangle, \langle e2,r_x,e1\rangle$

As we mentioned before, our ACC should have the capability to process large KGs. Our algorithm of ACC is designed to target this objective. Generally, the roles of TBox and ABox are different and so are their logic operations. TBox operations are based more on inferring and tracing or verifying class memberships in the hierarchy. ABox operations are more rule-based and govern fact checking, instance checking, consistency checking, and the like [110]. In a Knowledge Graph, the size of the ABox is usually much bigger than the size of TBox[1], so ABox reasoning is generally more complex on a larger scale than that for the TBox [110]. In our ACC strategy, the reasoning is performed on TBox and ABox separately. The TBox transformation and TBox IJP calculation relies on an OWL 2 reasoner. Usually the reasoning services are expensive [110], but our TBox reasoning is performed only once, and can be performed offline. The ABox scanning is the more active part of ACC, which is performed on the fly to match the IJPs in the ABox. This two-step strategy performs complex and time-consuming reasoning on a relatively small TBox, and performs simple scanning and matching on the large ABox. As a result, the ACC is efficient in dealing with large KGs, which allows us to integrate it in an iterative pipeline for identifying schema-inconsistent subsets or make use of it in a KGE training procedure for schema-aware negative sampling.

3.4 Schema-aware Hybrid Iterative Knowledge Graph Completion

We combine an E-method, a R-method, a M-method, a L-method and an ACC module to produce schema-correct triples. Similar to the E-methods, the L-methods also learn embeddings for entities and relations. In SICKLE, the difference of the L-methods and the E-methods is that, for L-methods, the entity or relation embedding is initialised by encoding its text literal with a pre-trained language model, while for E-methods, it is a one hotindex vector.

In a SICKLE pipeline, the ACC module acts as a "cleanup" step. The ACC is applied at the beginning of the pipeline to filter out any schema-inconsistent triples existing in the original KG and only schema-correct triples are fed to triple producers. Moreover, the outputs of the E-methods, the L-methods and the R-methods contain a large portion of triples that not consistent with the input KG; the ACC is run after each of these producer modules to identify a schema-correct subset and a schema-inconsistent subset, so that only the schema-correct subset are fed to next triple producer or next iteration for training and prediction. The schema-inconsistent triples are preferred to be selected as negative examples in our schema-aware negative sampling strategy. We do not run ACC after the M-method, because the triples from the M-method are already schema-correct.

---

1. Based on the LOD-a-lot survey of the Linked Open Data cloud, Frank van Harmelen estimates that of 23.8 billion unique statements only 565 million could be classified as rules - the rest being facts, i.e., rules make up just under 2% of the total. For more detail, see `https://frankvanharmelen.home.blog/2020/07/13/2-makes-all-the-difference-on-the-lod-cloud/` accessed 24.02.22.

In SICKLE, the schema-aware KGC only expands the input KG with schema-correct triples. The new version of KG $\mathscr{G}'$ consists of the original schema and schema-correct subset: $\mathscr{G}' = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{S}_{cor}$.

We assemble KGC pipelines with single or multiple triple producers and the ACC module. A pipeline can be executed in an iterative manner, and run in either series or parallel mode. An iterative pipeline can be run many rounds, until a certain stop condition is satisfied, for example, the specified number of iterations has been completed. In the series mode, each triple producer in a pipeline is executed one by one. The ACC is executed after each triple producer if needed, and only schema-correct triples are fed to next triple producer in the pipeline. In the parallel mode, all triple producers in a pipeline are executed in parallel. And after all triple producers finish prediction, their results are collected and merged, then fed to the ACC module. We illustrate examples of the series pipeline and parallel pipeline in Figure 3.1 and Figure 3.2.

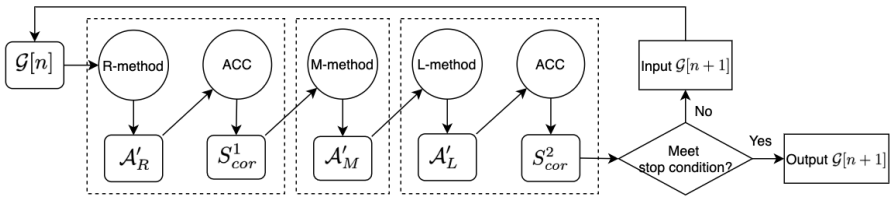

Figure 3.1: An example of series iterative pipeline with R-methods, M-method and L-methods.

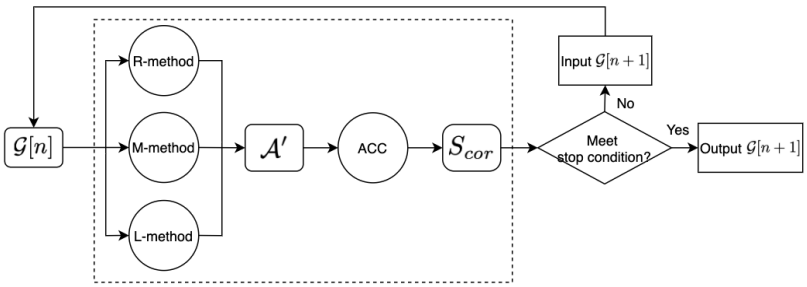

Figure 3.2: An example of parallel iterative pipeline with R-methods, M-method and L-methods.

In the combined and iterative pipeline, the new schema-correct subsets are incorporated into training and producing procedures. The KGEs learned from a combined and iterative pipeline are trained with more schema-correct triples from different triple producers and from iterations. We hope this data augmentation would contain more schema-related features, hence can produce more schema-correct triples than KGEs learned in single-pass and can bring benefits to downstream schema-related tasks, such as TP. SICKLE supports flexible

assembly of pipelines. By testing different pipeline combinations and configurations, we would like to compare pipeline performance in producing schema-correct triples. We also test the KGE learned from different pipelines on a downstream TP task, which exploits the KGE to predict entity types.

3.5 Schema-aware Negative Sampling

The embedding-based KGC approaches focus on learning low-dimensional embeddings for triple prediction. Conventional KGE methods, such as TransE [52], ComplEx [95], SimplE [94] and many the others, are trained through discriminating positive samples from negative ones. The sampling strategy, especially negative sampling strategy, is important for KGE training [166]. The CWA assumes that triples present in the KG are true and triples not present in the KG are false. A commonly used silver standard sampling strategy uses the existing triples in a KG as positive examples, and creates negative examples by corrupting positive triples and replacing head or tail with randomly selected entities [64]. However, this does not work under the open-world assumption, because absent triples could possibly be false negatives [65]. The negative sampling must consider the open world assumption, that the missing information is unknown rather than false. Also, KGs are often constructed based on knowledge extraction and may contain errors, hence existing triples may also be false positives. For example, about 10% of triples in the NELL-995 dataset are not consistent with its schema. In SICKLE, we use schema-correct triples as positive examples and leverage the ACC module to generate schema-inconsistent triples as negative examples during KGE training procedure.

Our schema-aware sampling strategy involves two subsets of an expanded ABox $\mathscr{A}'$: *the schema-inconsistent subset* ($\mathscr{S}_{incon}$) and *the schema-correct subset* ($\mathscr{S}_{cor}$). Based on Definition 2 - 5, these two sets are computed by equation (3.1) and (3.2), respectively.

$\mathscr{S}_{incon}$ contains the triples involved in any proof of false $\bot$ in the extended ABox $\mathscr{A}'$ but which do not occur in the input ABox $\mathscr{A}$. Here $\mathscr{G}' = \mathscr{A}' \cup \mathscr{T}$; function $\dot{\in}(\pi, \mathscr{G})$ returns the triple $r(h,t)$ that completes the proof π and $r(h,t) \notin \mathscr{A}$; $\mathscr{G}' \vdash_\pi \bot$ means that π is a proof of $\bot$ in $\mathscr{G}'$.

$$\mathscr{S}_{incon} = \{r(h,t) \mid \exists \pi.\ r(h,t) \dot{\in} (\pi, \mathscr{G}) \wedge (\mathscr{G}' \vdash_\pi \bot)\} \tag{3.1}$$

It's worth noting, the equation (3.1) can apply to the original KG, if we regard the original ABox as $\mathscr{A}'$ and $\emptyset$ as $\mathscr{A}$, any triples in the original ABox that complete the proof π are inconsistent triples under this equation. While for an expanded KG, we identify new triples that complete the proof π as inconsistent triples, but considering the interaction among all axioms, which is often the reason for inconsistency.

The set of schema-correct triples $\mathscr{S}_{cor}$ are the ones not in the schema-inconsistent subset $\mathscr{S}_{incon}$ and also satisfy the constraints $\mathscr{C}$, which are domain and range constraints in our case.

$$\mathscr{S}_{cor} = \{r(h,t) \mid r(h,t) \in (\mathscr{A}' \setminus \mathscr{S}_{incon}) \wedge C_h \equiv D_r \wedge C_t \equiv R_r\} \tag{3.2}$$

For positive sampling, we alter the traditional silver standard sampling by setting schema-correct subset $\mathscr{S}_{cor}$ as positive examples, instead of $\mathscr{A}$. We define the schema-aware positive sampling as below:

Definition 7 (Schema-aware Positive Sampling)**.** *Given a Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}'$, we use the schema-correct subset $\mathscr{S}_{cor}$ in $\mathscr{A}'$ as positive examples for triple producers in SICKLE.*

Our negative sampling strategy has two sources of negative samples: (1) a schema-inconsistent subset of top ranks in the last round prediction in an iterative pipeline; (2) randomly generated inconsistent triples. Intuitively, incorporating feedback from inconsistent predictions back to the iterative pipeline would reduce frequently encountered inconsistent predictions in future training. However, there's usually not enough top ranked inconsistent predictions from the last round to do all the negative sampling, especially when the specified number of negative samples is much more than the positive samples in the KGE model training. In that case, we need to dynamically generate inconsistent triples for each batch of training data loaded in the KG embedding training. The final schema-aware negative samples are selected from both the inconsistent top ranked predictions and inconsistent random generations. In practice, as the KG embedding training procedure reads training data in batches, we dynamically generate schema-inconsistent triples for each batch of positive examples B_{pos}. We formalize the generation of inconsistent negative examples as Definition 8:

Definition 8 (Random Schema-inconsistent Negative Examples)**.** *During the training procedure of a KGE model, given the positive examples $\mathscr{S}_{cor}$ and the schema $\mathscr{T}$, for each batch of triples $B_{pos} \subseteq \mathscr{S}_{cor}$, we randomly generate a set of schema-inconsistent negative examples:*

$$B_{incon} = \{r(h',t') \mid \mathscr{T} \cup \mathscr{S}_{cor} \models r(h',t') \sqsubseteq \bot\} \tag{3.3}$$

where $h' \in E_B$ and $t' \in E_B$, E_B are the entities in B_{incon}.

Example 4. *Given two relation assertions: α_1, α_2, which are in $\mathscr{A}'$ but not $\mathscr{A}$, and the TBox and ABox: $\mathscr{T} \cup \mathscr{A}'$, and the schema-correct subset of the ABox: $\mathscr{S}_{cor}$, where:*

1. $\mathscr{S}_{cor} \sqsubseteq \mathscr{A}'$
2. $\{\alpha_1, \alpha_2\} \sqcap \mathscr{S}_{cor} \sqsubseteq \bot$,
3. $\mathscr{T} \cup \mathscr{S}_{cor} \models \{\alpha_1, \alpha_2\} \sqsubseteq \bot$,
4. $(\mathscr{T} \cup \mathscr{S}_{cor} \cup \alpha_1)$ *is consistent,*
5. $(\mathscr{T} \cup \mathscr{S}_{cor} \cup \alpha_2)$ *is consistent.*

In this example, we can conclude that α_1 and α_1 belong to $\mathscr{S}_{incon}$. Thus, We cannot add $\{\alpha_1, \alpha_2\}$ into the target KG, because they complete a proof of inconsistency, with regard to item 3 in Example 4. However, neither α_1 or α_1 belongs to B_{incon}, with regard to item 4 and 5, because we only focus on the interactions between negative examples and existing positive examples when reasoning inconsistency for negative sampling.

In an iterative pipeline, we make use of the schema-inconsistent subset $\mathscr{S}_{incon}$ of top ranks from previous round predictions and select them as negative samples together with the generated inconsistent negative examples.

With $\mathscr{S}_{incon}$ and B_{incon}, we formalize the schema-aware negative sampling as Definition 9:

Definition 9 (Schema-aware Negative Sampling)**.** *During the training procedure of an KGE model, the schema-aware negative examples B_{neg} for batch B_{pos} are randomly sampled from $(\mathscr{S}_{incon} \cup B_{incon})$.*

The definitions in this section formalised a few key steps of algorithms in Section 3.7.2 and Section 3.3. Also, $\mathscr{S}_{cor}$ computed by equation (3.2) are incorporated in the evaluation in Section 3.8.

3.6 Schema-aware Silver Standard

A traditional evaluation strategy for KGC is to use a subset of the given KG as a test set, often referred to as silver standard evaluation. A problem with the silver standard approach is that a Knowledge Graph itself might not be complete, thus could potentially produce false negative results. As for gold standard evaluations, the result quality is usually measured in recall, precision, and F-measure [1]. The silver standard evaluation is usually applied to measure the performance of KGC approaches on how well a given triple is replicated by a KGC method, with Hits@n and MRR calculated against the test set [1].

We would like to produce more schema-correct triples. We also would like to see how well the triple producer elicits the schema-related features, for example, predicting implicit triples that are inferred by the given KG. Hence, we designed the *Schema-aware Silver Standard*. Instead of splitting a subset of a given KG as a test set, our schema-aware silver standard is tested on a union of the KG's materialised triple set and the test set split from a given KG. If only results of the M-method were used for schema-aware silver, the number of test triples might not be sufficient. Therefore, we combined results of the M-method and the test set split from the given KG as the schema-aware silver test set. We formalise the schema-aware silver standard for a given Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$ as below:

- **Schema-aware silver standard for LP** We split the relation assertions S_{rel} in $\mathscr{A}$ to $LP_{train}, LP_{dev}, LP^*_{test}$, the schema-aware silver test set S_{test} is generated by extend LP_{test} with relation assertions inferred from the M-method:
  $LP_{test} = LP^*_{test} \cup \{r(a,b) \mid \mathscr{G} \models r(a,b), r \in N_P, a, b, \in N_I\}$

- **Schema-aware silver standard for TP** We extend the type assertions S_{type} in $\mathscr{A}$ with materialised type assertions:
  $S'_{type} = S_{type} \cup \{A(a) \mid \mathscr{G} \models A(a), A \in N_C, a \in N_I\}$. the schema-aware silver standard for TP is generated by splitting the type assertions S'_{type} to $TP_{train}, TP_{dev}, TP_{test}$

N_C, N_P, N_I are named classes, named properties and instances in $\mathscr{G}$.

We evaluate LP and TP separately. We calculate Hits@n and MRR for LP evaluation. While for the TP, we calculate recall, precision, and F1, because the type assertions are relatively more complete than the relation assertions with regard to our datasets. Also the materialisation infers implicit entity type assertions and flattens the class hierarchy.

3.7 Implementation

3.7.1 Overall Architecture

Figure 3.3 illustrates the overall architecture of SICKLE. In this figure, the four triple producers in light grey colour are off-the-shelf packages that integrated as part of SICKLE system. We implemented all other modules from scratch. We assemble pipelines with four types of triple

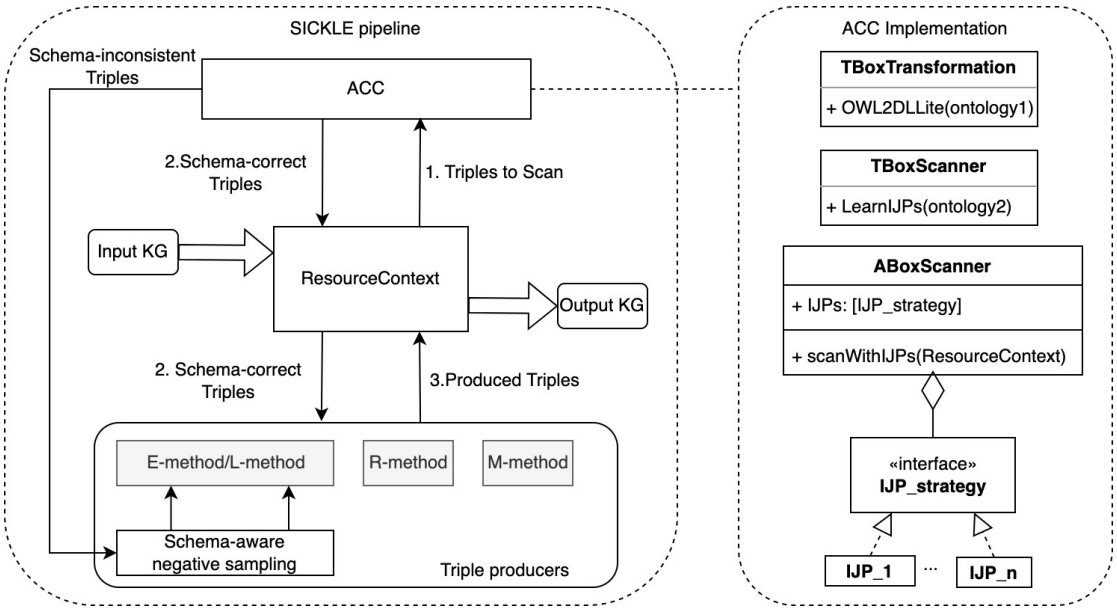

Figure 3.3: Overall structure of SICKLE and an example of ACC implementation.

producers: an E-method, a R-method, a L-method and an M-method and an ABox scanner from the ACC module. The E-methods and L-methods train KGEs, then use the KGEs to predict new triples based on the existing KG. We pick the top K triples from the ranked candidates, within a threshold γ. We integrated a literal based KGE system, namely BLP [87], and extend it to E-methods and L-methods. In the inductive mode, the BLP system encodes

an entity by encoding its textual description with a pre-trained language model, then the basic KG embedding learning algorithm is carried out as usual. It can be regarded as fine-tuning a pre-trained language model with an LP objective. BLP provides a convenient configuration to run inductive LP in combination with pre-trained language models, such as BERT [90]; and it supports a few relational models, namely TransE [52], ComplEx [95], and SimplE [94]. BLP also supports transduction LP, hence it can be executed as the E-methods as well. In order to fairly compare the experimental results for the E-methods and the L-methods, we used BLP as the experimental model of both the E-methods and the L-methods.

We adopted the inductive KGE learning algorithm in BLP as described in Algorithm 1, which makes use of a pre-trained language model for learning representations of entities via a LP objective. We used the pre-trained bert-base-cased model as the encoder, but other pre-trained models based on Transformers are equally applicable. Given an entity description $d_{ei} = (w_1, ..., w_n)$, the encoder pre-processes it by adding special tokens $[CLS]$ and $[SEP]$ to the beginning and end of the description. The format of the input to the language model is $\hat{d_{ei}} = ([CLS], w_1, ..., w_n, [SEP])$. The output is a sequence of contextual embeddings of the language model's hidden size for this piece of text, and we use representation of the $[CLS]$ token from the last layer of the BERT model as entity representations in our models.

Algorithm 1: Learning KGE via L-method

Input: KG $\mathscr{G}' = (N_I, N_P, Tr, \mathscr{D})$, entity encoder f_θ with parameters θ, and a KGE model ℓ

Output: θ

$\theta = \{\theta\} \cup \{r_j \mid r \in N_P\}$

foreach $(e_i, r_j, e_k) \in Tr$ **do**

  $(e'_i, r_j, e'_k) \leftarrow negativeSampling(e_i, r_j, e_k)$

  $s_p \leftarrow s(f_\theta(d_{ei}), r_j, f_\theta(d_{ek}))$

  $s_n \leftarrow s(f_\theta(d_{ei'}), r_j, f_\theta(d_{ek'}))$

  $\theta \leftarrow \ell(\theta, s_p, s_n)$

end

We implemented the TP as a multi-label classifier with the KGE learned from either E-methods or L-methods as input features. We only predict the maximum 50 most frequent types. The multi-label classification boils down to doing binary classification for each type; we use binary cross entropy to measure the error for each type. In the implementation, we combined the binary cross entropy loss function with a Sigmoid function. We used the same approach to calculate the loss of prediction.

We integrated AnyBURL [163], a popular rule-based LP model as base model of R-method. AnyBURL (acronym for Anytime Bottom-Up Rule Learning) treats each training fact as a compact representation of a very specific rule; it then tries to generalize it, in order to cover and satisfy as many training facts as possible. It has been analyzed in a recent paper [48], that AnyBURL is a very competitive KGC system in its accuracy and efficiency.

Materialisation, or the M-method, is a powerful tool for completing a KG. It uses a reasoner to compute all individuals' concepts and roles in a KG. We assume the original TBox is in OWL 2, which is a rich Description Logic. Our baseline system SIC used HermiT [131] to materialise KGs, however the materialisation on the NELL-995 dataset with HermiT took more than 24 hours on a 3.60GHz Intel i7-6850K CPU and 64G memory Linux server. It becomes urgent to speed up the M-method, so that we can integrate M-method in a KGC pipeline. Hence, SICKLE integrates a high-performance tractable OWL 2 reasoners, namely Konclude [92]. We extended and wrapped Konclude's interface, so that we can infer and query all entailed relation assertions and type assertions in one call. With Konclude, the materialisation of the NELL-995 dataset took less than 10 minutes. Therefor, it is feasible to integrate the M-method into a combined KGC pipeline.

The ACC module has three functions, the TBox transformation, the TBox scanner, and the ABox scanner. The TBox transformation is to approximate a TBox from OWl 2 to DL-Lite. The TBox scanner is to calculate IJPs from the TBox in DL-Lite. The TBox transformation, the TBox scanner are calculated only once in advance. We only need to integrate the ABox scanner in a pipeline. SICKLE assumes that a target Knowledge Graph has a schema, which at least contains subsumption hierarchies, as well as domain and range information. However, for Knowledge Graphs that lack a schema, one can use ontology learning tools [4] to generate a schema for such a Knowledge Graph. The details of ACC implementation are described in Section 3.7.3.

The new triples from the E-methods, the L-methods and the R-methods should be scanned with ACC, before merging into the existing KG. But the triples from the M-method are always schema-correct, the new relation assertion triples from M-method are directly added to a target KG.

The original implementations of individual KGC methods are scattered, making it difficult to combine them together to obtain the benefits of each. Users have to set up each system and spend extra time converting data formats that are required by different systems and transferring data between each other. To tackle this issue, we encapsulate the data format of triple producers. Each of them shares the same data interface and its triple producing process can be regarded as a black box. All triple producers and the ABox scanner use a context object to manage input and output data in a pipeline, so that it is convenient to combine them in a pipeline. The encapsulation has a few key benefits, including hiding data, more flexibility and

being easy to reuse. We only need to call the triple producer modules in just a few lines of code and there is no need to worry about the complicated dependencies, configurations and various data formats. With the encapsulation, we can assemble pipelines with multiple triple producers in one combination.

An assembled pipeline can be executed in an iterative manner, and the triple producers in a pipeline iteration can be executed in series or in parallel. All these execution options are configured at running time.

3.7.2 Schema-aware Sampling Strategy

There are two aspects to our schema-aware sampling strategy: (1) schema-correct triples as positive samples; (2) schema-inconsistent triples as negative samples. In SICKLE, the schema-aware positive sampling is that we only feed schema-correct triples to individual triple producers, and it is mandatory in our KGC pipeline. The schema-aware negative sampling means that we generate and select schema-inconsistent triples as negative samples in KG embedding training procedure. The schema-aware negative sampling is an optional feature in our KGC pipeline. Both positive and negative sampling strategies rely on ACC module, but use it in different ways.

When we detect schema-correct triples in a pipeline, the input of ACC is the whole KG $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}'$, SICKLE uses schema $\mathscr{T} \cup \mathscr{C}$ to supplement a set of schema-correct triples for the data sub-graph $\mathscr{A}'$. The data sub-graph $\mathscr{A}'$ could be the initial ABox or an expanded ABox.

The schema-aware negative sampling is performed in the KG embedding training procedure. The KG embedding model training procedure reads training data in batches, hence we dynamically generate schema-aware negative samples for each batch of training data. We describe the process of schema-aware negative sampling in Algorithm 2: In step 2, we generate a number of random negative triples for each positive triple. $B_{corrupt}$ may contain false negative triples, based on the open-world assumption. Hence, we identify the schema-inconsistent subset from $B_{corrupt}$, according to Definition 8. In step 5, we randomly select negative examples from both the random schema-inconsistent subset and the schema-inconsistent subset of previous predictions.

Algorithm 2: Schema-aware Negative Sampling

Input:
TBox: $\mathscr{T}$,
schema-correct set: $\mathscr{S}_{cor}$,
schema-inconsistent set: $\mathscr{S}_{incon}$,
a minibatch of size b from $\mathscr{S}_{cor}$: B_{pos}
Output: (B_{pos}, B_{neg})

foreach $r(h,t) \in B_{pos}$ **do**
    // sample n corrupted triples
    $B_{corrupt} \leftarrow B_{corrupt} \cup corrupt\big(r(h,t), n\big)$;
end
// Definition 8
$B_{incon} \leftarrow identifyBatchInconsistency(\mathscr{T}, \mathscr{S}_{cor}, B_{corrupt})$;
// Definition 9
$B_{neg} \leftarrow randomSample(B_{incon} \cup \mathscr{S}_{incon})$;

3.7.3 Algorithms of Approximated Consistency Checking

According to the approach described in Section 3.3, the implementation of ACC consists of three parts:

- **TBox Transformation**: a TBox approximation module replaces the TBox $\mathscr{T}_1$ of a source ontology in a more expressive DL $\mathscr{L}_1$ (OWL 2) with its least upper-bound $\mathscr{T}_2$ in a less expressive DL $\mathscr{L}_2$ (DL-Lite).
- **TBox IJP Calculation**: a TBox scanner module infers $\mathscr{T}_2$ and calculates the ABox and TBox IJPs.
- **ABox IJP scanning**: An ABox scanner module scans ABox $\mathscr{A}$ and filters out triples that match the IJPs learned in step 2.

TBox Transformation We apply an idea of knowledge compilation by semantically approximating a source ontology $\mathscr{O}_1$ in a more expressive DL $\mathscr{L}_1$ (source language OWL 2) with its least upper-bound $\mathscr{O}_2$ in a less expressive DL $\mathscr{L}_2$ (target language DL-Lite). In this setting, we have $\mathscr{O}_1 \models \mathscr{O}_2$ [167]. In our strategy, we only apply knowledge compilation on the TBox.

In this thesis, we have designed an algorithm to transform TBox from OWL 2 to DL-Lite, so that we can transform most of the OWL 2 ontology syntax to DL-Lite syntax.

The algorithm of “TBox Transformation from OWL 2 to DL-Lite" is described in Algorithm 3. We assume the source TBox $\mathscr{T}_1$ is in OWL 2. Following [167], we use N_C, N_P to denote the set of named classes and named properties used in $\mathscr{T}_1$. For named properties in $\mathscr{T}_1$, we assign new named classes D to class existential description $\exists r$ and $\exists r^-$. We assign new named classes N to the complement expressions of named classes $N_C \cup D$. The basic idea is to represent those expressions with their name assignments, so that the classification service

Algorithm 3: TBox transformation from OWL2 to DL-Lite

Input: TBox in OWL2: $\mathscr{T}_1$
Output: TBox in DL-Lite: $\mathscr{T}_2$

Initialize $\mathscr{T}_2 \leftarrow \emptyset$, $N \leftarrow \emptyset$, $D \leftarrow \emptyset$;
foreach $A_i \in N_C$ **do**
    $N_i \leftarrow \neg A_i$;
    $N \leftarrow N \cup \{N_i\}$
end
foreach $r_k \in N_P$ **do**
    $D_k \leftarrow \exists r_k$;
    $D_{k+1} \leftarrow \exists r_k^-$;
    $D \leftarrow D \cup \{D_k, D_{k+1}\}$;
    $N_k \leftarrow \neg D_k$;
    $N_{k+1} \leftarrow \neg D_{k+1}$;
    $N \leftarrow N \cup \{N_k, N_{k+1}\}$;
end
$\mathscr{T}_2 \leftarrow \mathscr{T}_1 \cup D \cup N$;
// call a $\mathscr{L}_1$ reasoner to infer $\mathscr{T}_2$
$Classification(\mathscr{T}_2)$;
foreach $r_k \in N_P$ **do**
    Replace N_k with $\neg D_k$ in $\mathscr{T}_2$;
    Replace N_{k+1} with $\neg D_{k+1}$ in $\mathscr{T}_2$;
    Replace D_k with $\exists r_k$ in $\mathscr{T}_2$;
    Replace D_{k+1} with $\exists r_k^-$ in $\mathscr{T}_2$;
end
foreach $A_i \in N_C$ **do**
    Replace N_i with $\neg A_i$ in $\mathscr{T}_2$;
end

from an OWL 2 reasoner can normalize them to subsumption form.

$$\begin{aligned} D &= \{D_k \equiv \exists r_k, D_{k+1} \equiv \exists r_k^-\} \\ N &= \{N_k \equiv \neg D_k, N_{k+1} \equiv \neg D_{k+1}, N_i \equiv \neg A_i\} \end{aligned} \tag{3.4}$$

where r_k denotes the named properties in N_P, A_i denotes the named classes in N_C, $1 \leq k \leq num_r$, num_r is the total number of named properties, $1 \leq i \leq num_a$, num_a is the total number of named classes in $\mathscr{T}_1$.

Then we apply the classification service of an OWL 2 reasoner, which computes all the subsumption among named classes and named properties in the expanded TBox: $\mathscr{T}_1 \cup D \cup N$. After applying classification, the class axioms in the expanded TBox are normalised into the following two forms: $B_1 \sqsubseteq B_2$ and $B_1 \sqsubseteq \neg B_2$, including axioms using vocabulary in D and N.

Then we replace D and N with their original expressions in those axioms having vocabulary of D and N. The purpose of replacing is to remove the additional named classes D and N, while keeping their axioms in the form of class expressions.

In Algorithm 3, we use the classification service provided by the TrOWL reasoner, but slightly extended the interface of TrOWL so that we get all the direct and indirect class subsumption and object property subsumption in one call.

TBox IJP Calculation We design and calculate IJPs based on the approximated TBox containing DL-Lite expressions. We kept the *Asymmetric* and *Inreflexive* constraints from OWL 2, because they are not expressed in DL-Lite, but we need them in IJPs. The optimised IJPs are described in Table 3.1. Algorithm 4 illustrates how to compute pattern 1 in Table 3.1 by TBox reasoning.

Algorithm 4: TBox IJPs 1, $\exists r \sqcap A \subseteq \bot$

Input: TBox $\mathscr{T}_2$

Output: $p_1 = \{(r, Set_A) \mid r \in N_P, Set_A = \{A \mid \exists r \sqcap A \sqsubseteq \bot, A \in N_C\}\}$, where N_P are named properties, N_C are named classes in $\mathscr{T}_2$

```
Initialize P_1 ← ∅ ;
foreach r ∈ N_P do
    Set_A = ∅ ;
    foreach ∃r ⊑ ¬A_i in T_2 do
        Set_A ← Set_A ∪ {A_i} ;
    end
    Add (r, Set_A) into p_1 ;
end
```

Algorithm 5: TBox IJPs 2, $\exists r1 \sqcap \exists r2 \subseteq \bot$

Input: TBox $\mathscr{T}_2$

Output: $p_2 = \{(r1, Set_{r2}) \mid r1 \in N_P, Set_{r2} = \{r2 \mid \exists r1 \sqcap \exists r2 \sqsubseteq \bot\}\}$, where N_P are named properties in $\mathscr{T}_2$;

```
Initialize P_2 ← ∅ ;
foreach r1 ∈ N_P do
    Set_r2 = ∅ ;
    foreach ∃r1 ⊑ ¬∃r2 in T_2 do
        Set_r2 ← Set_r2 ∪ {r2} ;
    end
    Add (r1, Set_r2) into p_2 ;
end
```

The TBox transformation and The TBox scanner are implemented in Java with a tractable OWL 2 reasoner named TrOWL [135]. TrOWL inherits the OWLAPI [2], which is more convenient for us to manipulate the ontology.

2. http://owlapi.sourceforge.net/

ABox IJP scanning Once we learned the IJPs from the TBox, we scan the ABox to identify inconsistent triples that complete the IJPs in column 3 of Table 3.1. The inconsistency can be repaired by removing any of the assertions that complete the patterns. In our iterative pipeline setting, we prefer to remove newly predicted triples, although they may not be the smallest subset of repairing.

Algorithm 6: ABox Scanning Strategy of p_1

Input: IJP: p_1, ABox subset to scan: $\mathscr{S}$
Output: Schema-inconsistent subset $\mathscr{S}_{incon1}$ under p_1

Initialize $\mathscr{S}_{incon1} \leftarrow \emptyset$, $\mathscr{S}_{con1} \leftarrow \mathscr{S}$;
foreach $(r, Set_A) \in p_1$ **do**
  $TR_r \leftarrow getTriplesByProperty(r)$;
  foreach $\langle h,r,t \rangle$ *in* TR_r **do**
    if $isNew(\langle h,r,t \rangle)$ *and* $getRDFTypes(h) \cap Set_A \neq \perp$ **then**
      $\mathscr{S}_{incon1} \leftarrow \mathscr{S}_{incon1} \cup \{\langle h,r,t \rangle\}$;
    end
  end
end

Algorithm 7: ABox Scanning Strategy of p_2

Input: IJP: p_2, ABox subset to scan: $\mathscr{S}$
Output: Schema-inconsistent subset $\mathscr{S}_{incon2}$ under p_2

Initialize $\mathscr{S}_{incon2} \leftarrow \emptyset$, $\mathscr{S}_{con2} \leftarrow \mathscr{S}$;
foreach $(r1, Set_{r2}) \in p_2$ **do**
  $TR_{r1} \leftarrow getTriplesByProperty(r1)$;
  $TR_{r2} \leftarrow getTriplesByProperties(Set_{r2})$;
  $Heads_{r2} \leftarrow getAllHeadEntities(TR_{r2})$;
  foreach $\langle h,r,t \rangle$ *in* TR_{r1} **do**
    if $isNew(\langle h,r,t \rangle)$ *and* $h \in Heads_{r2}$ **then**
      $\mathscr{S}_{incon2} \leftarrow \mathscr{S}_{incon2} \cup \{\langle h,r,t \rangle\}$;
    end
  end
end

Because the TBox IJPs focus on property characteristics, the triples fed to the individual pattern scanning strategy are grouped by properties. Algorithm 6 illustrates how to scan the ABox with pattern 1 of Table 3.1, which checks whether any classes of the head entity belong to the disjoint set of domain. Algorithm 7 illustrates how to detect inconsistent subsets with pattern 2 of Table 3.1, which has two relation assertions to complete the proof of inconsistency. We only check new triples in an expanded ABox, but consider the interaction of all axioms. If the input ABox is the initial KG, we treat all relation assertions as new triples, and check consistency for all triples. For each IJP, we scan the ABox assertions once to $(num_r - 1) * num_a$

Algorithm 8: ABox Scanner for consistency checking

Input: IJPs $P = [p_1, p_2, ..., p_n]$, ABox $\mathscr{A}$, expanded ABox $\mathscr{A}'$
Output: Schema-consistent subset $\mathscr{S}_{con}$ and schema-inconsistent subset $\mathscr{S}_{incon}$ under P, where $\mathscr{S}_{con} \cup \mathscr{S}_{incon} = \mathscr{A}'$ and $\mathscr{S}_{con} \cap \mathscr{S}_{incon} = \bot$

Initialize $\mathscr{S}_{incon} \leftarrow \emptyset$, $\mathscr{S}_{con} \leftarrow \emptyset$;
foreach p_i *in* P **do**
    $\mathscr{S} \leftarrow A' \setminus \mathscr{S}_{incon}$;
    `// identify inconsistent triples in` $\mathscr{S}$ `with` p_i
    $\mathscr{S}_{incon} \leftarrow \mathscr{S}_{incon} \cup scanWithIJP(p_i, \mathscr{S})$;
end
$\mathscr{S}_{con} \leftarrow \mathscr{A}' \setminus \mathscr{S}_{incon}$;

Algorithm 9: ABox Scanning for domain and range constraints

Input: Schema-consistent subset $\mathscr{S}_{con}$
Output: Schema-correct subset $\mathscr{S}_{cor}$ that satisfy domain and range constraints.

Initialize $\mathscr{S}_{con} \leftarrow \emptyset$;
foreach $r \in N_P$ **do**
    $TR_r \leftarrow getTriplesByProperty(r)$;
    foreach $\langle h, r, t \rangle$ *in* TR_r **do**
        if $getRDFTypes(h) \cap getDomain(r) \neq \bot$ *and* $getRDFTypes(t) \cap getRange(r) \neq \bot$ **then**
            $\mathscr{S}_{cor} \leftarrow \mathscr{S}_{cor} \cup \{\langle h, r, t \rangle\}$;
        end
    end
end

times, with regard to different IJPs, where num_r is the number of relations and num_a is the number of ABox assertions. Hence, the ABox scanning is polynomial-time complete. In our implementation, the ABox scanning is further accelerated by a data analysis tool, such as Pandas [3] and its GPU extension CuDF [4].

Each IJP in Table 3.1 has an ABox scanner strategy that implements the same interface, so that we can flexibly register specific IJPs in a pipeline. Algorithm 8 shows the program of consistency checking for an ABox, with regard to the equation (3.1) and the equation (3.2).

The schema-correct subset is finally computed by scanning the schema-consistent relation assertions to identify those satisfying domain and range constraints as described in Algorithm 9.

3. https://pandas.pydata.org/
4. https://rapids.ai/start.html

Our ACC approach is a soundness-preserving syntactic approximation. Since the IJPs listed in Table 3.1 do not cover all possible inconsistent patterns, it cannot guarantee soundness for inconsistency checking and justifications. Hence, we make ACC fully extensible to future variants, where all IJP strategies have the same interface but design their own scanning strategies, as shown in Figure 3.3.

3.8 Results and Analysis

3.8.1 Evaluation Matrix

Schema-aware Silver Completeness Ratios

We evaluated a variety of schema-aware KGC pipelines with single methods or their combinations in the correctness ratio, coverage ratio, consistency ratio and their harmonic mean.

The function $f_{Correctness}$ is to calculate the schema correctness ratio of a KGC approach across all iterations.

$$f_{Correctness} = \frac{1}{n} * \sum_{i=1}^{n} \frac{\mid \mathscr{S}_{cor}^{i} \mid}{\mid \varepsilon_i \mid} \tag{3.5}$$

where ε_i is the set of new triples produced by a triple producer at iteration i, $\mathscr{S}_{cor}^{i}$ is the schema-correct triples in ε_i.
Without knowing the 'complete' Knowledge Graph, it is rather hard to define a proper measure for completeness. Instead, [3] used the notion of coverage, to account for the scale of schema-correct triples added into the original Knowledge Graph. The function $f_{Coverage}$ is defined as:

$$f_{Coverage} = \frac{\sum_{i=1}^{n} \mid \mathscr{S}_{cor}^{i} \mid}{\mid \mathscr{Y} \mid} \tag{3.6}$$

where $\mathscr{Y}$ is a subset of the target Knowledge Graph $\mathscr{G}$ that consists of schema-unknown plus schema-correct triples. Larger $f_{Coverage}$ scores are an indication that the used KGC pipeline has produced a higher number of new schema-correct triples.

We define $f_{Consistent}$ as:

$$f_{Consistency} = \frac{1}{n} * \sum_{i=1}^{n} \frac{\mid \mathscr{S}_{con}^{i} \mid}{\mid \varepsilon_i \mid} \tag{3.7}$$

where $\mathscr{S}_{con}^{i}$ is the schema-consistent triples in ε_i.
The function f_h is the harmonic mean of $f_{Correctness}$, $f_{Coverage}$ and $f_{Consistent}$.

$$f_h = \frac{3}{\frac{1}{f_{correctness}} + \frac{1}{f_{coverage}} + \frac{1}{f_{Consistency}}} \tag{3.8}$$

These functions are to evaluate how good the KGC pipelines are at producing schema-correct triples.

Schema-aware Silver Standard

A traditional evaluation strategy for KGC is to use a subset of the given KG as a test set, often referred to as silver standard evaluation. A problem with the silver standard approach is that a Knowledge Graph itself might not be complete, thus could potentially produce false negative results. As for gold standard evaluations, the result quality is usually measured in recall, precision, and F-measure [1]. The silver standard evaluation is usually applied to measure the performance of KGC approaches on how well a given triple is replicated by a KGC method, with Hits@n and MRR calculated against the test set [1].

In this thesis, we would like to produce more schema-correct triples. We also would like to see how well the triple producer elicits the schema-related features, for example, predicting implicit triples that are inferred by the given KG. Hence, we designed the *Schema-aware Silver Standard*. Instead of splitting off a subset of a given KG as a test set, our schema-aware silver standard is tested on a union of the KG's materialised triple set and the test set split from a given KG. If only results of the M-method were used for schema-aware silver, the number of test triples might not be sufficient. Therefore, we combined results of the M-method and the test set split from the given KG as the schema-aware silver test set. We formalise the schema-aware silver standard for a given Knowledge Graph $\mathscr{G} = (\mathscr{T} \cup \mathscr{C}) \cup \mathscr{A}$ as below:

- **Schema-aware silver standard for LP** We split the relation assertions S_{rel} in $\mathscr{A}$ to $LP_{train}, LP_{dev}, LP^*_{test}$, the schema-aware silver test set S_{test} is generated by extend LP_{test} with relation assertions inferred from the M-method:
  $LP_{test} = LP^*_{test} \cup \{r(a,b) \mid \mathscr{G} \models r(a,b), r \in N_P, a, b, \in N_I\}$
- **Schema-aware silver standard for TP** We extend the type assertions S_{type} in $\mathscr{A}$ with materialised type assertions:
  $S'_{type} = S_{type} \cup \{A(a) \mid \mathscr{G} \models A(a), A \in N_C, a \in N_I\}$. the schema-aware silver standard for TP is generated by splitting the type assertions S'_{type} to $TP_{train}, TP_{dev}, TP_{test}$

N_C, N_P, N_I are named classes, named properties and instances in $\mathscr{G}$.

We evaluate LP and TP separately. We calculate Hits@n and MRR for LP evaluation. While for the TP, we calculate recall, precision, and F1, because the type assertions are relatively more complete than the relation assertions with regard to our datasets. Also the materialisation infers implicit entity type assertions and flattens class hierarchy.

3.8.2 Datasets

Our notion of schema-consistency in this thesis relies on 2 criteria: (1) high enough number of triples, and (2) a schema that has a rigid hierarchy of concepts and rich object properties. Following the dataset analysis in [3], we performed experiments with three Knowledge Graphs: the NELL-995 Knowledge Graph, the DBpedia politics subset, namely DBped-P [3].

The NELL-995 Knowledge Graph is a dataset developed at Carnegie Mellon University and contains 142,065 triples. The schema of the NELL-995 Knowledge Graph has 1187 concepts, 894 object properties and many types of object property axioms, such as inverse object properties, functional object property, asymmetric object property, irreflexive object property, and object property domain and range. DBpedia-2016 is a large-scale, multilingual Knowledge Graph that has more than 3 billion triples, 685 concepts and 2,795 properties. DBped-P is a subset of DBpedia-2016, containing 352,754 triples that are related to political issues. We use a subset of the original DBpedia-2016 schema [5], only contains the concepts, properties and their axioms related to the ABox assertions in DBped-P. It has 305 concepts, 188 object properties and many types of object property axioms, such as inverse object properties, functional object property, and object property domain and range. We also expanded the ABox of the original DBped-P with more type assertions from a recent DBpedia-2021 release.

Considering our notion of schema-correctness, we manually create constraints for those properties missing domain or range, by generating dummy domain and range constraints as the union of the types based on the types of head and tail entities. After the repairing, the initial schema-correctness ratios of DBped-P and NELL-995 are over 99%. [47] pointed out that SIC's efficacy is limited to the number of constraints furnished before running consistency checking. In Table 3.2, we compared the $f_{correctness}$ scores of top 10 predictions with TransE on both original and repaired datasets. Our schema-aware experiment is performed on the repaired datasets, which can better reflect the efficacy of different strategies described in this chapter.

Table 3.2: $f_{correctness}$ of top 10 predictions with E-methods on original and repaired datasets

Dataset	TransE		SimplE		ComplEx	
	Original	Repaired	Original	Repaired	Original	Repaired
NELL-995	0.34	0.62	0.33	0.61	0.31	0.54
DBped-P	0.43	0.64	0.45	0.63	0.41	0.67

5. http://wiki.dbpedia.org/downloads-2016-04

3.8.3 Experimental Setting

We assembled pipelines with four types of triple producers: the R-methods, the M-method, the E-methods and the L-methods. We chose BLP implementation as L-methods and the E-methods because its code is easy to extend. We were able to build additional functionality on top of it, such as different negative sampling methods and an extra text encoder. Under the E-method and L-method categories, we evaluated three KG embeddding models: TransE, SimplE and ComplEx which are supported by BLP. BLP supports a limited set of base models. TransE is well-known and representative in translation-based KGE category [48, 136]. SimplE and ComplEx are typical models under Bi-linear KGE category [48, 136]. In BLP, the L-methods and the E-methods share same implementation of loss-function and training objective, the only difference is the initial encoder of entities and relations, so that we are able to take a fair comparison between these two.

The literal features in the L-methods are entity names and descriptions. We used BERT as the text encoder in the L-method for both NELL-995 and DBped-P datasets. For the NELL-995 dataset only, we tested two different types of literal features: one is the original entity text, another is the type extended literal, which is the concatenation of entity type name and entity name. The type extended literal contains additional explicit type information. We did not test the other two datasets with type extended literal, because the literal features for DBped-P dataset were extracted from WikiPedia pages, and the corpus set for training the BERT pre-trained models contains WikiPedia pages. We assume that the pre-trained BERT models have already encoded some type context for DBped-P entities.

We ran each triple producer in either single-pass or 2 to 3 iterations, according to the sizes of datasets. Also, we ran pipelines with combined triple producers in iterative manner.

We evaluated individual methods and combined pipelines in both schema-aware silver completeness ratios and schema-aware silver standard matrix.

SIC is an initial effort in schema-aware KGC; however, it has not provided runnable source code, thus preventing us from comparing SICKLE with SIC.

3.8.4 Empirical Results

Schema-aware Silver Completeness ratios

We compare the schema-aware silver completeness ratios among a variety of pipelines, with regard to our research objectives and strategies described in this chapter.

In Figure 3.4 and 3.5, we compared the E-methods and the L-methods with iterative pipelines. In these three figures, we use the original embedding-based models as baseline and ran them in an iterative manner, without applying ACC at any middle point. With only the base model, the $f_{correctness}$ and $f_{consistency}$ dropped quickly in iterations, because both schema-correct,

schema-unknown and schema-incorrect triples are added into the target KG in iterations. With SICKLE pipelines, the E-methods and the L-methods achieved higher $f_{correctness}$, $f_{consistency}$ and f_h scores than the baselines in iterations. It means SICKLE pipelines are more efficient in producing schema-correct triples than the basic embedding models in the iterative KGC pipeline. This observation indicates that the ACC module played an important role in the SICKLE pipeline and improved the overall performance of producing schema-correct triples. In both Figure 3.4 and Figure 3.5, the L-methods outperformed the E-methods in almost all schema-aware silver completeness ratios. In Figure 3.4, the performance of the L-methods with extended type literals is even better. One possible reason is that higher-quality literal features, such as the type information, can positively impact the performance of L-methods.

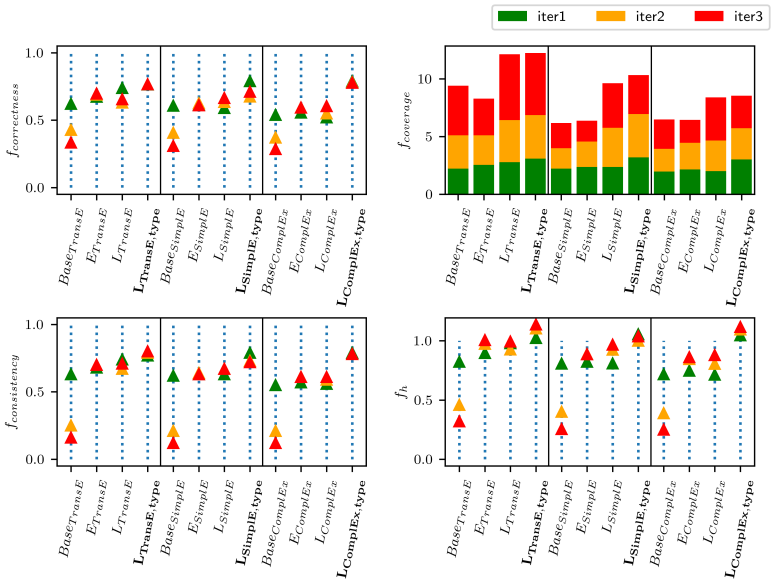

Figure 3.4: NELL-995 dataset. Compare E-methods and L-methods in the schema-aware silver completeness performance

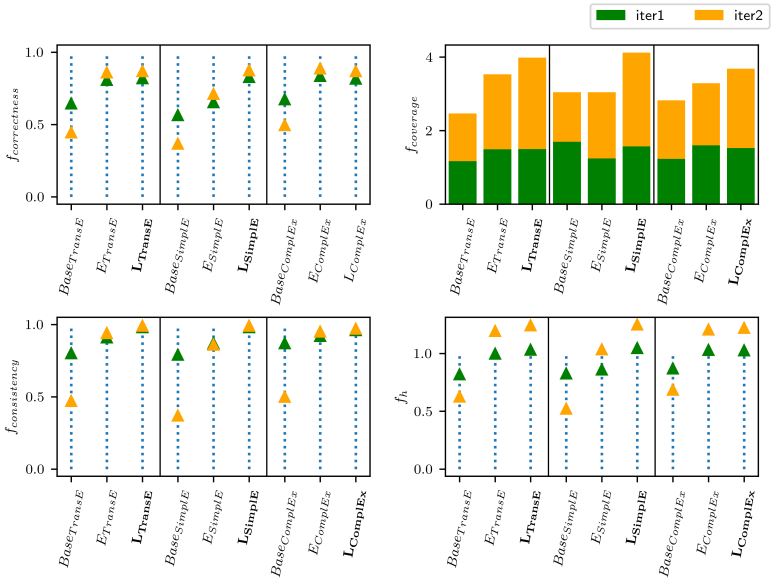

Figure 3.5: DBped-P dataset. Compare E-methods and L-methods in the schema-aware silver completeness performance

In Figure 3.6 and Figure 3.7, we compared the E-methods, the E-methods with schema-aware negative sampling, the L-methods, and the L-methods with schema-aware negative sampling. From these figures, we can see that the triple producers with schema-aware negative sampling outperformed the triple producers without it in all schema-aware silver completeness ratios. It means that the schema-aware negative sampling strategy improved the E-methods and the L-methods in producing schema-correct triples.

Although the schema-aware negative sampling strategy has a positive impact on the performance of schema-aware silver completeness ratios, involving ontological reasoning in the training procedure would significantly increase the computing resource consumed. The E-methods with schema-aware negative sampling strategy took more than 24 hours on the NELL-995 dataset for just one iteration, while it was about 2 hours without the schema-aware negative sampling strategy on a Linux server with an Intel Xeon Silver 4314 CPU and a Nvidia A100 80GB PCIe GPU. Hence, we only ran one iteration for NELL-995 and DBped-P datasets with schema-aware negative sampling strategy, because there should be a balance between the small incremental performance and the large resource and time consuming.

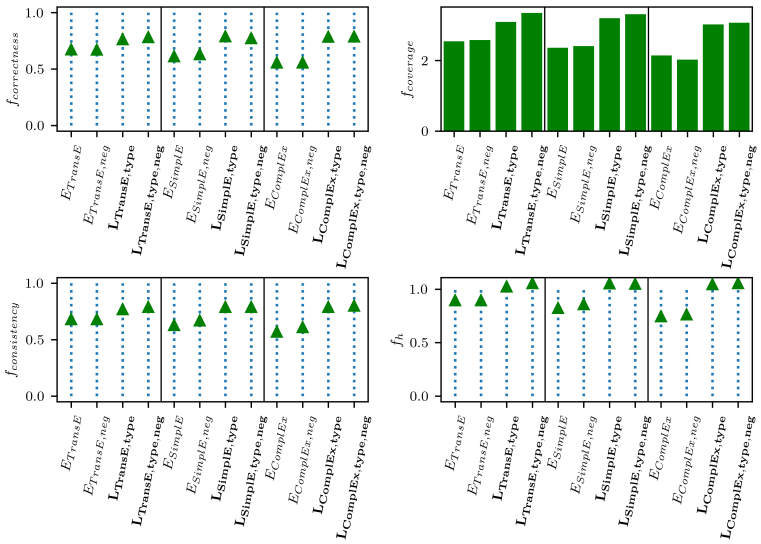

Figure 3.6: NELL-995 dataset. Compare E-method, L-method and schema-aware negative sampling strategy in the schema-aware silver completeness performance

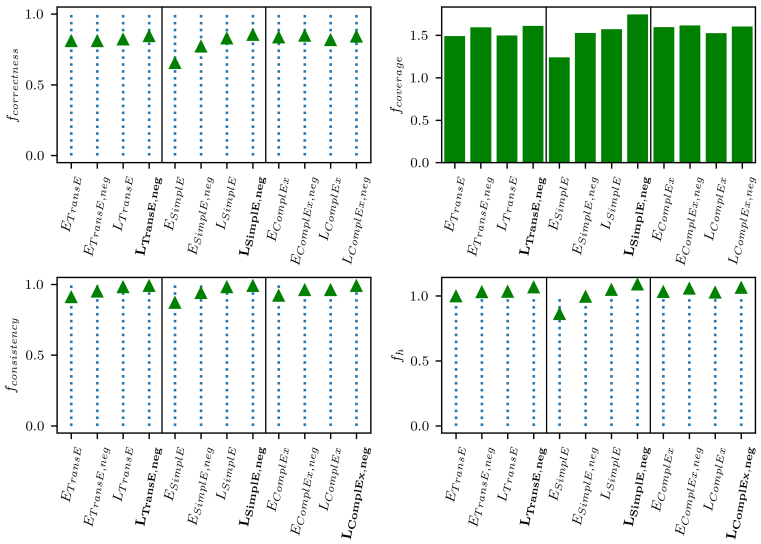

Figure 3.7: DBped-P dataset. Compare E-method, L-method and schema-aware negative sampling strategy in the schema-aware silver completeness performance

In Figures 3.8 and 3.9, we compared the M-method, the L-methods, the R-methods and their combinations. The M-method achieved highest scores in $f_{correctness}$ and $f_{consistency}$, but lowest scores in $f_{coverage}$. The R-methods outperformed the L-methods in all four measures, which means the R-methods is more capable of catching the schema features in predicting new triples. We also combine different triple producers in pipelines and run in an iterative manner.

The triple producers are combined and executed in series or in parallel. In Figure 3.8, the one round combinations outperformed most of the single methods running three rounds in $f_{coverage}$ with NELL-995 dataset. In Figure 3.9, the combined pipeline generally achieved higher scores in f_h. The series pipeline slightly outperformed the parallel pipeline in the first round, while as the number of iterations increases, this gap decreases. For obvious reasons, the series pipeline shares new data during each iteration, while the parallel pipeline does not share new data until the second iteration. The above observations indicate that combining different types of triple producers can produce more schema-correct triples.

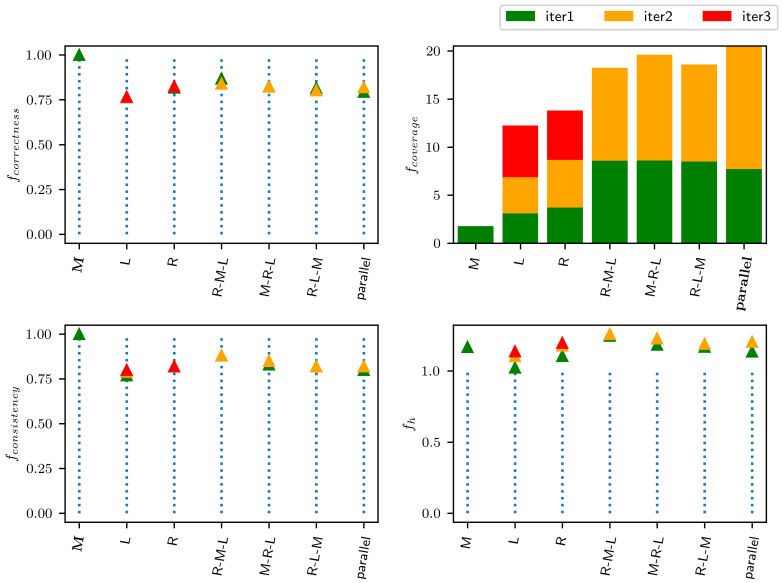

Figure 3.8: NELL-995 dataset. Compare single methods and combined methods in the schema-aware silver completeness performance

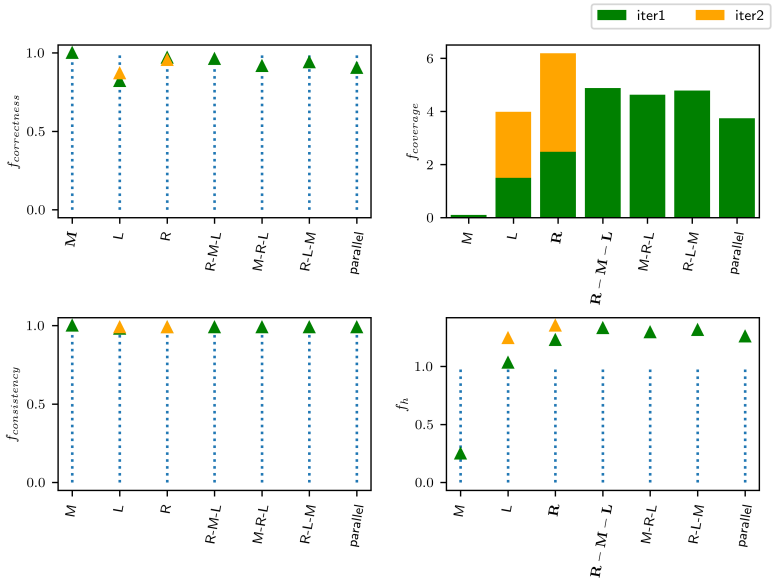

Figure 3.9: DBped-P dataset. Compare single methods and combined methods in the schema-aware silver completeness performance

We configured the embedding dimension for TransE, SimplE and ComplEx to 128 on two datasets. The learning rate, batch size and maximum training iterations were configured with regard to the size of the training set and the loss convergence. We ran the combination pipelines with DBped-P and NELL-995 on a Linux server with an Intel Xeon Silver 4314 CPU and a Nvidia A100 80GB PCIe GPU. The execution time of a combination pipeline for the NELL-995 dataset was about 3.5 hours in serial and 2.5 hours in parallel. The execution time of a combination pipeline with DBped-P was much longer, which was about 8 hours for 1 iteration in serial mode and 4 hours in parallel mode. Most of the execution time was spent in training KGE and the prediction with R-methods (The AnyBURL training was much faster than its prediction on large KGs). The execution time for a single method in 3 iterations was much longer than the combined pipeline in 1 iteration. We recommend running the pipeline in combination and in parallel mode, as it is the most efficient and effective mode.

We summarize the observations as below:

- The L-methods outperformed the E-methods in almost all schema-aware silver completeness ratios, given high quality literal features and text encoder. One possible reason is that higher-quality literal features, such as the type information, can be transferred to KG embedding via L-methods to some degree.

- The schema-aware negative sampling strategy slightly improved the schema-aware silver completeness ratios for the E-methods and the L-methods. If the E-methods are used as a benchmark to compare the L-methods and the schema-aware negative sampling strategy, then the growth from the schema-aware negative sampling strategy is not as significant as the effect of the literal features. Also, the L-methods are more efficient than E-methods with the schema-aware negative sampling strategy, as the later need extra time to do ontology reasoning during its training procedure.
- The R-methods achieved the best scores on $f_{coverage}$ and achieved relative high scores on all other schema-aware silver completeness ratios. It means the LP results of the R-methods are usually more schema-aware than the E-methods and the L-methods.
- The M-method produced triples of very good quality; however, its coverage level is far from that of the other triple producers. We recommend combining the M-method with other methods to produce more high-quality results.
- Combinations of different types of triple producers outperformed single triple producers in the production of schema-correct triples.

Schema-aware Silver Standard Matrix

We tested the R-method, the E-methods and the L-methods on both LP and TP tasks based on schema-aware silver standard. In the LP evaluation, we calculated Hits@1,3,10 and MRR. And we calculated recall, precision, and F1 in the TP evaluation for entities and their types.

We also compared two different negative sampling strategies on E-methods and L-methods: one is random negative sampling, another is schema-aware negative sampling. For NELL-995, we tested two different types of literal features for the L-method: one is the original entity text, another is the type extended literal, which is the concatenation of entity type name and entity text. We also evaluated the KGEs learned from combination pipelines in the downstream TP task. The results of schema-aware silver standard LP are described in Tables 3.3, 3.4, 4.5, and the results of TP are described in Tables 3.5, 3.6 and 4.6. In these tables, E denotes E-method, L denotes L-method, neg denotes schema-aware negative sampling, $type$ denotes L-method with additional type literal.

In Table 3.7, the KGEs fed to TP are learned from last L-method in the pipeline. We use the base embedding model having the highest TP scores in the combination pipeline. For NELL-995, we used the type extended literals for L-method.

Table 3.3: NELL dataset schema-aware silver standard evaluation in LP.

Model	Measure	E	L	L_{type}	E_{neg}	$L_{type,neg}$
TransE	Hits@1	0.04	0.05	0.08	0.04	0.08
	Hits@3	0.10	0.13	0.14	0.10	0.16
	Hits@10	0.18	0.23	0.24	0.20	0.26
	MRR	0.09	0.12	0.14	0.10	0.15
SimplE	Hits@1	0.17	0.02	0.10	**0.17**	0.08
	Hits@3	0.27	0.06	0.15	**0.28**	0.17
	Hits@10	0.39	0.18	0.25	**0.39**	0.28
	MRR	0.24	0.07	0.15	**0.24**	0.16
ComplEx	Hits@1	0.18	0.06	0.11	**0.18**	0.12
	Hits@3	0.28	0.11	0.18	**0.28**	0.19
	Hits@10	0.40	0.21	0.28	**0.41**	0.28
	MRR	0.25	0.11	0.17	**0.25**	0.18

Table 3.4: DBped-P datasets schema-aware silver standard evaluation in LP.

Model	Measure	E	L	E_{neg}	L_{neg}
TransE	Hits@1	0.03	0.06	0.07	0.05
	Hits@3	0.11	0.14	0.12	0.14
	Hits@10	0.21	0.27	0.23	0.24
	MRR	0.09	0.13	0.11	0.12
SimplE	Hits@1	0.08	0.06	**0.08**	0.06
	Hits@3	0.18	0.13	**0.19**	0.18
	Hits@10	0.28	0.23	**0.29**	0.24
	MRR	0.15	0.13	**0.16**	0.13
ComplEx	Hits@1	0.11	0.06	**0.10**	0.06
	Hits@3	0.21	0.13	**0.20**	0.14
	Hits@10	0.31	0.24	**0.33**	0.24
	MRR	0.18	0.12	**0.18**	0.12

Table 3.5: NELL dataset schema-aware silver standard evaluation in TP.

Model	Iteration	Measure	E	L	L_{type}	E_{neg}	L_{neg}	$L_{neg,type}$
TransE	1	Pr	0.79	0.77	0.87	0.79	0.82	0.84
		Rec	0.65	0.72	0.78	0.80	0.89	0.90
		F1	0.69	0.75	0.82	0.80	0.85	0.87
	3	Pr	0.79	0.90	**0.93**	-	-	-
		Rec	0.70	0.82	**0.80**	-	-	-
		F1	0.74	0.86	**0.86**	-	-	-
SimplE	1	Pr	0.66	0.63	0.72	0.75	0.77	0.78
		Rec	0.92	0.71	0.74	0.76	0.76	0.84
		F1	0.72	0.66	0.73	0.75	0.77	0.82
	3	Pr	0.75	0.79	**0.88**	-	-	-
		Rec	0.75	0.75	**0.81**	-	-	-
		F1	0.75	0.77	**0.84**	-	-	-
ComplEx	1	Pr	0.70	0.64	0.71	0.82	0.80	0.83
		Rec	0.68	0.89	0.76	0.85	0.83	0.85
		F1	0.69	0.70	0.73	0.83	0.81	0.84
	3	Pr	0.71	0.78	0.78	-	-	-
		Rec	0.71	0.73	0.78	-	-	-
		F1	0.71	0.75	0.78	-	-	-

Table 3.6: DBped-P schema-aware silver standard evaluation in TP.

Model	Iteration	Measure	E	L	E_{neg}	L_{neg}
TransE	1	Pr	0.86	0.86	0.87	0.85
		Rec	0.95	0.97	0.96	0.98
		F1	0.90	0.90	0.90	0.91
	2	Pr	0.86	**0.87**	-	-
		Rec	0.97	**0.99**	-	-
		F1	0.91	**0.92**	-	-
SimplE	1	Pr	0.85	0.86	0.85	0.86
		Rec	0.95	0.98	0.95	0.98
		F1	0.90	0.91	0.90	0.91
	2	Pr	0.90	**0.87**	-	-
		Rec	0.91	**0.99**	-	-
		F1	0.91	**0.92**	-	-
ComplEx	1	Pr	0.86	0.86	0.86	0.86
		Rec	0.96	0.96	0.98	0.98
		F1	0.90	0.91	0.90	0.91
	2	Pr	0.86	**0.87**	-	-
		Rec	0.97	**0.99**	-	-
		F1	0.91	**0.92**	-	-

Table 3.7: The schema-aware silver standard evaluation in TP, compared the L-methods and the combination pipeline of R-method, M-method and L-methods.

Dataset	Iteration	Measure	L	R-M-L
NELL	1	Pr	0.87	0.89
		Rec	0.78	0.88
		F1	0.82	0.89
	3	Pr	0.93	**0.94**
		Rec	0.80	**0.93**
		F1	0.86	**0.93**
DBped-P	1	Pr	0.86	0.86
		Rec	0.96	0.97
		F1	0.91	0.91
	2	Pr	0.87	-
		Rec	0.99	-
		F1	0.92	-

Based on these results, we made the following analysis:

E-methods VS L-methods

As described in Table 3.3 and 3.4, the E-methods significantly outperformed the L-methods in the LP task in most cases. We attribute this to the more challenging problem faced by literal encoders: they must learn a complicated function from words to an entity representation, while the E-methods learns a lookup table with one embedding per entity and relation. This observation is also reported in [87].

While as described in Table 3.5 and 3.6, the L-methods outperformed the E-methods in the TP task, given high quality literal and text encoders. In our experimental setting, the TP is a downstream task of LP. It uses the entity embeddings learned in LP as input feature to predict multiple entity types. The L-methods encode additional context features from literal, for example, the cosine similarity between *Barack Obama* and *George W. Bush* from BERT encoding is 0.997. And these additional features are transferable to the downstream TP task. This is especially reflected in the type extended literal applied to the NELL dataset: encoding type information with the entity literal can effectively improve LP and TP performance.

Schema-aware negative sampling strategy

The schema-aware negative sampling strategy brings positive effects in LP and TP with the DBped-P and NELL-995 datasets.

Although the schema-aware negative sampling strategy can improve the performance in LP and TP, it requires much more computing resources and time to reason each batch of training data, so as to generate schema-inconsistent negative samples. We did not apply the schema-aware negative sampling strategy to the iterative pipeline for all datasets, because we needed to consider the balance between performance and cost.

Single-pass, combination and iterative pipelines

We evaluated TP with single-pass, combination and iterative pipelines. Our hypothesis is that the embeddings learned in an iterative manner may have an advantage in the downstream TP task, because the schema features are enhanced by injecting new schema-correct data in iterative training. Our experimental results in Table 3.5 and 3.6 confirmed this hypothesis. Of the three datasets, the iterative learned KGE performs better than the single-pass KGE in the downstream TP task. Also, the TP performance of the L-methods increased more than the E-methods, with the increase of iteration. We further injected schema-correct data from the R-method and M-method in KG embedding training, by running combined pipelines in an iterative manner. In Table 3.7, the combined pipeline achieved higher scores than the single L-method. This indicates that the KGEs from the combined and iterative KGC pipeline encode more features related to the schema, and have the potential to benefit downstream task such as TP.

3.8.5 Comparing our ACC with existing Reasoners

We evaluate our ACC approach with a designed use case: given a KG, we extended it with a set of randomly generated relation assertions, then we collected all the justifications that were generated by a tractable reasoner. We checked whether our approach could detect all the incorrect relation assertions that are stated in each of these explanations. We compared our ACC approach with existing reasoners in terms of:

- How many inconsistent triples (in percentage) that were identified by an existing reasoner, can also be detected by our approach.
- How long our ACC module took compared to the reasoning times of existing reasoners.

In doing reasoning, the existing reasoners have two steps which are the consistency checking and generating justification, hence the total time for a reasoner is the consistency checking plus generating justification. Generating justification is often most costly since it needs to calculate the minimal inconsistent subset of a Knowledge Graph. Our ACC approach consists of three steps, which are TBox transformation, TBox scanning, and ABox scanning. The total time of our approach is the sum of three steps.

We show the effectiveness of our ACC approach, a complete and sound reasoner, namely HermiT [91], an approximate reasoner TrOWL and the ACC service provided by SIC. For both Hermit and TrOWL, we implemented the experiment with OWLAPI to verify consistency and generate explanations if the input KG is not consistent. The ACC service provided by SIC doesn't have TBox transformation, and it relies on HermiT to identify its IJPs in the original TBox. We used three datasets: DBped-P and NELL-995, but expanded the initial KG by adding a few randomly generated inconsistent triples. The results of comparison can be seen in Table 3.8 In Table 3.8, *Exp* stands for the number of explanations that the justification service of HermiT generated. Each explanation consists of one or several axioms that cause

Table 3.8: ACC Performance Comparison

Dataset	HermiT			TrOWL			SIC			Our Approach		
	Exp	CC	JG	Cov	CC	JG	Cov	TS	AS	Cov	TT	TS	AS
DBped-P	25	32s	3:50:00	100%	9s	2:39:20	100%	2:45:00	0:57:18	100%	0:14:30	5s	0:1:56
NELL	12	5s	2:10:14	100%	2s	1:40:23	100%	1:15:00	0:40:30	100%	4:10:31	16s	0:2:07

inconsistency in the input KG. *CC* stands for the time that the reasoner took for "Consistency Checking" and *JG* stands for the time for "Justification Generation". *Cov* stands for the coverage of the explanations. It refers to how many axioms in the explanations detected by the HermiT's justification service were also identified by TrOWL, SIC, and our approach. *TT* stands for "TBox Transformation". Our TBox transformation is the most costly among these three steps, since it needs to calculate all subsumption in named classes and properties and the extended named classes described in Algorithm 3. But the TBox transformation is a one-off process, and only needs to be calculated once in advance. *TS* stands for "TBox scanning" and *AS* stands for "ABox scanning". Both the TS and the AS are much more efficient than the conventional reasoning service for generating justification. Our implementation is more efficient and quicker than SIC's TS and AS.

3.9 Summary

In this thesis, we revisited the notion of the schema-aware Knowledge Graph completion problem and presented a schema-aware iterative hybrid Knowledge Graph completion system, namely SICKLE. SICKLE provides a combinational implementation that allows easy assembling of four types of triple producers (embedding-based method, literal-embedding based method, rule-based method and materialisation) in pipelines and performs approximate consistency checking on produced triples, so that only schema-correct triples are added to the target KG.

Previous research [93] shows that translation based methods, such as TransE, cannot properly capture simple rules; even Bi-linear models, such as SimplE and ComplEx, are severely limited when representing subsumption or equivalence between relations. This indicates that embedding based triple producers might have limited capability in terms of representing schemas of Knowledge Graphs. We tackle this problem by considering a few strategies from different level. For the system level, we combined different types of triple producers and an approximated consistency checking module to produce schema-correct triples and ran it in an iterative manner. Only schema-correct triples are added to the target KG and used in iterative training. From a functional level, we integrated literal-embedding based methods. We found that the knowledge from pre-trained language models improves schema-awareness in KGC tasks, in particular, benefit schema-related KGC tasks, such as TP. At the algorithm level, we implemented a schema-aware sampling strategy for the E-methods and the L-methods. The schema-correct triples are sampled as positive examples and schema-inconsistent triples are sampled as negative examples. We found that this sampling method had a positive effect in producing schema-correct triples in pipeline and improved the performance in downstream TP task.

Chapter 4

Enriching a Log KG, an Application of SICKLE in Real Scenario

4.1 Background

Knowledge Graphs could facilitate operation and maintenance (O&M) activities and eventually reduce its overall cost [101, 102, 119]. Most often, O&M KGs are manually constructed by domain experts, which is problematic because 1) they only have limited coverage and do not provide sufficient information to support further analysis, e.g. root cause analysis and 2) given that the network systems are changing over time, previously acquired knowledge could become outdated, but once the networks were deployed, we have limited access to inspect the networks and update the knowledge. A network company is looking for methods to overcome these problems, and find out a solution that can reduce the O&M cost. Hence, the TREAT (Tacit Relation Extraction and Transformation) project was proposed and is aimed to use the product manual and system logs to automatically construct an ever-evolving knowledge base for the 5G network Product.

Different from traditional knowledge extraction that extracts triples from free text, our extraction is based on the 5G network domain. Our source of knowledge includes technical documents, expert knowledge and especially logs. Logs are a vital source of information for monitoring software systems' running status and diagnosing faults. For instance, Example 5 describes an error message from an service-oriented system, showing the status that one service failed to request another service. The background knowledge is that these two types of services are for certain functions and one depends on another.

Example 5. *[ERROR][2021-10-07 20:48:15.347 +08:00][197.28.1.23] [Session-database-container-0se45][UserExecSvc-098]: "Failed to request UserDomainSvc-034, unreachable"]*

Given that logs are weakly structured, vary in different formats, and do not follow natural language grammar, we may need some novel techniques to extract knowledge from logs. Also, because the knowledge extraction approaches are error-prone, it is necessary to have some mechanisms to identify and remove the faulty knowledge. As the knowledge graph

evolves, the process of completion becomes crucial, and that is where the SICKLE system takes place. Then we can make the evolution of the knowledge graph an iterative process by combining multiple knowledge graph completion (KGC) methods to enrich the KG. The growth of knowledge graphs makes automatic validation increasingly impractical.

Our objective in this chapter is to build a logic-based Knowledge model on multiple resources (network logs, technical documents and expert knowledge), to facilitate knowledge presentation and reasoning, including construction and inferring additional knowledge from multiple resources and applying constraints on learned knowledge. There are a few challenges in this task:

- Because our data sources are weakly linked and raw log messages are usually unstructured and contain a lot of sub-words, acronyms and terminologies, we cannot leverage general pre-trained language models to tokenize and transform the raw log messages into vectors for extraction steps, such as named entity recognizing against normal text.
- Due to the accumulative nature of log data, in the real world, the knowledge grows as logs come on stream; as a result, reasoning [168, 169] covering a whole KG is more and more resource consuming over time. It is necessary to keep the reasoning within a manageable query space.
- Some existing log analysis approaches [102] only check if a reference entity exists, yet checking logical or semantic consistencies of the extracted knowledge has not been well addressed.

To tackle these challenges, we combine a variety of NLP methods and logic rule-based extraction methods together to extract and infer knowledge from multiple data sources, and we apply constraint-based triple validation on new triples to guarantee that the new triples do not conflict with known rules. Then, we apply the SICKLE system on the extracted KG and learn more schema-consistent triples.

4.2 Problem Statement

In this chapter, we address the third research question by exploring an approach to automated knowledge graph construction and enrichment from a 5G network's system logs.
Q3: In a real 5G network scenario, given multiple complex data sources, how to make the a domain specific KG an evolving process by combining data-driven methods and knowledge-based methods, with regard to the schema of KG?

The designed enrichment process consists of two steps: extraction and completion. During the KG construction and completion process, we pay close attention to the issue of correctness. These sub-tasks are interrelated and are key tasks for knowledge representation and reasoning.

The first step is to extract knowledge from logs. The second step treats the enrichment process as a KGC problem. The SICKLE system combines an embedding-based method, a literal embedding-based method, a rule-based method and materialization to complete the extracted KG from the extraction step. It learns new triples in an iterative manner, making it possible to be applied on this real dynamically growing KG. The reason that we separate the overall process into two steps is that usually the log extraction task depends on dynamic incoming data source, while the iterative KGC task is more flexible and can be scheduled at any time.

In this chapter, we firstly present a log extraction approach for log knowledge graph (KG) construction. It includes a novel strategy that utilizes inference rules from a background knowledge graph to learn new triples and validate triples. Also, it implements a local to global strategy to perform reasoning on temporary log instance graphs (LIGs) then on the extended background knowledge graph (BKG), which significantly reduces the query space. We call the log extraction system LEKG (Log Extraction Knowledge Graph). To explain how the LEKG system extracts and infers triples, we demonstrate the applicability of this approach by a use case in the context of root cause analysis. Using the LEKG system, we extracted a dataset called TREAT.

At the second stage, we used the SICKLE system to learn more schema-correct triples, and evaluate SICKLE's Schema-aware Silver Completeness Ratios on the TREAT dataset. The TREAT KG possesses different textual features, such as entity names and descriptions in customised vocabulary derived from logs, compared to general KGs like DBpedia and NELL995. We adapted SICKLE to the TREAT KG by replacing the BERT encoder with a language model specifically trained from logs. We would like to see how SICKLE's multiple KGC assemble works on this real scenario KG, and how well it produces schema-correct triples in a combined pipeline.

4.3 How do we combine NLP and logic rules together to construct a KG from system logs

Background knowledge, such as internal expert knowledge, architectural information and external knowledge [101, 102, 119], exists in real enterprise scenarios. For example, BKGs can be obtained by information extraction or exported from other data sources. The data sources include technical documents, expert knowledge and especially logs. We constructed a background knowledge graph by applying classic information extraction techniques such as semantic parsing on the technical manual to produce a set of high level conceptual triples. We also converted some expert knowledge to RDF triples and included them in the background

knowledge graph. Unlike existing works [101, 102, 119, 120] that only use BKGs to align and map new entities, our view is that the background knowledge contains not only entities but also logical rules that describe the logical connections and constraints between modules of the 5G network.

In our work, new triples are learned from the KG by rule inference and validated by constraints from background knowledge. Only valid triples are included in the enriched KG. Example 6 is a positive rule describing the connection between a container and a service. Example 7 is a negative rule describing the constraints of dependencies between two services, which means no dependency loop in the system.

Example 6. $Container1(?s) \wedge Service1(?o) \wedge hosts\_service(?s, ?o) \rightarrow depends\_on(?s, ?o)$

Example 7. $Service1(?s) \wedge Service2(?o) \wedge depends\_on(?s, ?o) \wedge depends\_on(?o, ?s) \rightarrow \bot$

To this end, we developed a system (LEKG) that extracts and learns knowledge from logs. To keep the reasoning to amanageable query space, we designed a local to global strategy to infer triples. We first construct temporary LIGs based on groups of logs having specific features, then perform reasoning against each LIG for inferring relations between log instance entities. After merging triples from LIGs into the BKG, we perform reasoning on this global BKG to infer implicit relations. Then we apply constraint-based triple validation on new triples. Finally, we describe a use case that uses the LEKG for root cause analysis. The contributions of LEKG are summarized as below:

1. A novel strategy to extract knowledge from large volumes of logs;
2. A system, called LEKG, to integrate extraction, rule-based relation linking, constraint-based validation;
3. A use case that applies LEKG to root cause analysis.

The combination of background knowledge and a huge number of logs is a typical scenario in information technology companies. Hence we formulate our research problem 1 as follows.

Definition 10 (Aim of Construction)**.** *Given a BKG $G_{\mathbb{B}}$, a set of logs in their original format $\mathbb{L}$ and a log theory $\tau = R_{positive} \cup R_{negative}$, where $R_{positive}$ are positive rules that infer new triples and $R_{negative}$ are negative rules that are constraints for validation, the task is to update BKG $\mathbb{G}$ by extracting knowledge from $\mathbb{L}$ to enrich $G_{\mathbb{B}}$.*

4.4 Methodology of LEKG: log extraction that combines NLP and logic rule inference

The structure of our log extraction system is illustrated in Figure 4.1.

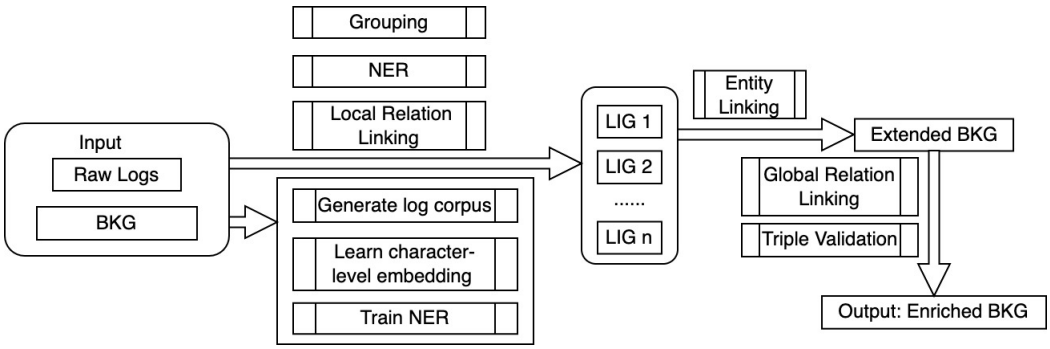

Figure 4.1: Process of Log Extraction

The automated log parsing converts free text to structured information. However, the parsed items usually only have vague categories; entities and events may hide in the body of parsed items. Hence, we combine Named Entity Recognition (NER) [170] and templates to identify entities from log messages. By training a domain-specific NER model, we identify and label the entities based on their semantic context. The entity labels are assigned to entities as RDF types and contribute to relation linking in downstream steps.

Instead of simply merging entities to BKG, we use Horn rules to infer relations. A Horn rule is a disjunction of atoms with at most one unnegated atom. In the implication form, they have the following format: $A_1 \wedge A_2 \wedge ... \wedge A_n \rightarrow B$, where $A_1 \wedge A_2 \wedge ... \wedge A_n$ is the body of the rule and B is the head. A positive rule has an atom as its head, while a negative rule has the head $\perp$. Thus, a negative rule is in the form of $A_1 \wedge A_2 \wedge ... \wedge A_n \rightarrow \perp$. Positive Horn rules can help to generate new triples, and negative rules can help to identify contradicting triples [72, 123].

In the real world, the knowledge grows as logs come on stream; as a result, reasoning [168, 169] covering a whole KG is more and more resource consuming over time. To this end, we apply a local to global strategy for inferring relations. Firstly, we aggregate logs by their source components. We believe the logs from the same source have more clustered interactions between them. Secondly, we construct LIGs using entities identified in aggregated log groups, after which rule inference over R_{local} is applied to each LIG. Then, the entities and triples are learned from each LIG locally.

Definition 11 (Local Reasoning)**.** *Given a set of logs $\mathbb{L}$ divided into n groups $g_1, \ldots g_n$ of logs based on their sources, where $g_i (1 \leq i \leq n)$ is a group of logs from the same source, a subset $R_{local} \subseteq R_{positive}$ of local positive rules , a function f_g extracting a LIG from $g_i \in \mathbb{L}$, and the entailment closure operation Cn. The set of local logical consequences T_{g_i} of each g_i can be computed as follows.*

$$T_{g_i} = Cn(f_g(g_i) \cup R_{local}) \tag{4.1}$$

Entities from different LIGs may have potential relationships, but are not linked within each local reasoning process. Hence, we apply reasoning with R_{global} against the whole extended BKG to infer implicit relations for all entities.

Definition 12 (Global Reasoning)**.** *Given an original BKG $G_{\mathbb{B}}$, local consequences T_{g_i}s, a subset $R_{global} \subseteq R_{positive}$ of global reasoning rules,and the entailment closure operation Cn. The extended BKG $G'_{\mathbb{B}}$ is computed as follows:*

$$G'_{\mathbb{B}} = Cn(G_{\mathbb{B}} \cup \{T_{g_i} | 1 \leq i \leq n\} \cup R_{global}) \tag{4.2}$$

Although we perform relation linking by rules, there is no guarantee that these Horn rules are sound. Even if rules are sound for the current system, the conditions may change while the system evolves. Thus we validate new triples by a set of constraints and the triples that do not pass the validation are called negative triples. The triples from the input graphs $G_{\mathbb{B}}$ are not negative because $G_{\mathbb{B}}$ is from the last version and has passed its validation.

Definition 13 (Validation)**.** *The set of negative triples $\mathbb{N}_{triples}$ are ones involved in any proof of false $\perp$ in the extended graphs $G'_{\mathbb{B}}$ but which do not occur in the input $G_{\mathbb{B}}$. Here $\mathbb{T} = G'_{\mathbb{B}} \cup R_{global} \cup R_{negative}$ and $\mathbb{T} \vdash_{\pi} \perp$ means that π is a proof of $\perp$ in $\mathbb{T}$.*

$$\mathbb{N}_{triples} = \{r(s,o) | \exists \pi.\ r(s,o) \dot{\in} (\pi, G_{\mathbb{B}}) \wedge \mathbb{T} \vdash_{\pi} \perp\} \tag{4.3}$$

where function $\dot{\in}(\pi, G_{\mathbb{B}})$ returns the last incoming triple $r(s,o)$ that completes the proof π and $r(s,o) \notin G_{\mathbb{B}}$.

The new version of graph $\mathbb{G}$ is output after filtering out $\mathbb{N}_{triples}$.

Definition 14 (Filter)**.** *The final KG $\mathbb{G}$ is computed by removing negative triples $\mathbb{N}_{triples}$ from the extended graphs $G'_{\mathbb{B}}$.*

$$\mathbb{G} = G'_{\mathbb{B}} \setminus \mathbb{N}_{triples} \tag{4.4}$$

4.5 LEKG Implementation

4.5.1 Entity Extraction

Instead of purely relying on templates, we combine the template-based approach and the NER approach to identify and label entities from log messages. Because raw log messages are usually unstructured and contain a lot of sub-words, acronyms and terminologies, we cannot leverage general pre-trained language models to tokenize and transform the raw log messages into vectors for NER. Hence we first generate a log text corpus by log parsing and learn character-level representation for log messages via FastText[1]. The word vector

1. https://fasttext.cc/

generated through FastText via N-Gram technique holds extra information about sub-words. We annotate the log corpus using entities from BKG and use the spaCy[2] toolkit to train its NER model. We combine Regex-based NER with the trainable NER model, because some entities like IP addresses are more suitable to be identified by Regex. The NER is performed on each log instance.

4.5.2 LIG Construction

We generate two types of graphs, one is the BKG, and the other is the LIGs. Initially the BKG holds concept level triples that are learned from technical manuals and converted from expert knowledge. The LIGs are temporary graphs consisting of entities identified from groups of logs. Triples learned from LIGs will be fused to the BKG, then the LIGs will be deleted.

Though the logs are weakly structured, we still recognised some patterns to help construct the LIGs. We firstly group the logs by multiple fixed fields like “containers" and “components", which are the source modules to produce logs. In each individual log group, the entities are identified and divided into two sets, one is of self-contained entities identified in the grouping keys, another is of entities identified in the variable log messages and we are not sure if they belong to the source module. Each entity is assigned *rdf:type* properties by its NER label and an additional class *SELFCONTAINED* indicating whether it is identified from the source components. For example, given this log from a service-oriented system:

```
[ERROR][2021-10-07 20:48:15.347 +08:00][197.28.1.23]
[Session-database-container-0se45][UserExecSvc-098]:
"Failed to request UserDomainSvc-034, unreachable"]
```

Container instance entity *Session-database-container-0se45* and service instance entity *UserExecSvc-098* are the source container and source service that generate this log, and service instance entity *UserDomainSvc-034* is likely an external entity that interacted with *UserExecSvc-098*.

We link entities within each LIG group. The positive rules for relation linking in LIGs are designed based on the entity classes. An example positive rule to infer relation *host_service* is:

$$\begin{aligned} &Container(?s) \wedge Service(?o) \wedge \\ &SELFCONTAINED(?s) \wedge SELFCONTAINED(?o) \\ &\quad \rightarrow host\_service(?s, ?o) \end{aligned}$$

2. https://spacy.io/

4.5.3 Link Entities to BKG

Linking and aligning entities from LIGs to the BKG is a key step that facilitates the enrichment of the BKG. We cannot simply merge the triples from LIG to BKG, because they have different identifiers and the former have classes such as SELFCONTAINED, that we would not like to include in the BKG. Different from typical entity linking task that aims to link text to entities in a KG, our entity linking module has two functions: 1) link entities from the LIGs to the BKG based on text and NER labels; 2) link instance entities from LIGs to corresponding concepts in the BKG. For example, entity *Session-database-container-0* is an instance of class *Session-Database-Container*. We would like to link instance entities with their concept entities, so that we can learn concept level triples from instance triples or link instance entities to each other based on their conceptual dependency. The entity linking module is a combination of SPARQL query [3], ElasticSearch [4] and character-level text similarity comparison. We implement the entity linking module to support case-insensitive, sub-word matching and context-aware, so that it has the ability to apply both exact matches and fuzzy matches. If no existing entity is found in BKG, we create a new entity and assign *rdf:type* as its label.

4.5.4 Rule-based Relation Linking

We apply positive rules to infer new triples in both LIG and BKG. The rules for LIG and BKG are different. The former links entities in an LIG with explicit relations; while the latter infers implicit relations for both concept entities and instance entities.

We manually scripted such positive rules to involve domain knowledge in the BKG. This positive rule set could possibly be extended by rule mining tools such as AMIE [124–126] and Rudik [72, 123]. We use the Pellet reasoner [134] to infer triples. BKG reasoning happens after the set of LIG triples are merged into BKG.

4.5.5 Triple Validation

Given a new triple $r(s,o)$ and an example negative rule such as:

$$r(?s,?o) \wedge r'(?s,?o) \rightarrow \perp$$

If $r'(s,o)$ exists in BKG, then $r(s,o)$ is invalid.

The negative rules are converted from BKG's ontology or scripted manually to involve expert knowledge. There are different types of negative rules, such as class disjointness $C_1(?x) \wedge C_2(?x) \rightarrow \perp$, relation restriction $r(?a,?b) \wedge r(?b,?a) \rightarrow \perp$ and rules that restricts the system's behaviours: $r1(?a,?b) \wedge r2(?a,?c) \rightarrow \perp$.

---

3. https://www.w3.org/2001/sw/wiki/SPARQL
4. https://www.elastic.co/elasticsearch/

Practically, we parse the negative rules into SPARQL queries and check whether there exist any contradictions in the BKG against the new triple.

4.6 A Use Case of Log Extraction and Logic Inference for Root Cause Analysis

We tested our proposed approach on a set of logs generated by a testing 5G network of Huawei. The service-oriented architecture breaks the system logic into different, small services where each one has a single task or responsibility. The services are hosted and executed inside containers such as Docker [5]. The different services communicate and cooperate with each other to provide the system functionality as a whole. Our test data were logs exported from an enterprise operation and management platform. In practice, the local to global strategy reduces the reasoning query space. Since reasoning is segmented within a set of LIGs, only a subset of inference rules are applied to the whole BKG to infer implicit relations between entities from different LIGs. Also, the reasoning in LIGs can be performed in parallel to save execution time. Our use case demonstrated (1) how to learn knowledge from logs and (2) how the knowledge learned from logs facilitates root cause analysis.

4.6.1 Extracting Knowledge from Logs

A full set of log groups is too large to demonstrate in this report, therefore we use a single log instance as an example and pretend it is a log group having only one log instance. The extraction process is illustrated in Figure 4.2. The process is described as below:

1. The NER identifies a list of entities from the log.
2. The entities and labels are converted into RDF triples as $\{1,2,3\}$. Three entities are linked to concepts in BKG: $\{4,5,6\}$. Two entities are assigned with rdf:type SELFCONTAINED as they are in fixed columns: $\{7,8\}$. These triples compose an initial LIG.
3. Apply the set of positive rules on this LIG, and learn a set of triples as $\{9,10\}$. The local rule $R1$ infers triple 9 and $R2$ infers triple 10.
4. Align and merge triples from LIG to the background graph. Extend the background graph with triples $\{1,2,3,4,5,6,9,10\}$.
5. Infer new triples based on all triples in the background graph. 0 is the triple from the initial background graph, $\{5,6,9\}$ are from the above LIG, $\{11,12,13,14\}$ are from other LIGs.
6. New triple 15 is learned from the extended BKG, based on triples $\{0,5,6,9,12,14\}$ and rule $R3$.

5. https://www.docker.com/

Figure 4.2: A use case of extracted knowledge for root cause analysis.

After extraction and inference, we expand the BKG with a set of instance level triples $\{1,2,3,4,5,6,9,10\}$ from the log and a triple 15 that connects two alarm instance entities.

4.6.2 Use the Log Knowledge Graph for Root Cause Analysis

The background knowledge now contains initial concept level triples and triples learned from a set of LIGs. In Figure 4.2, triples $\{12,13,14,15\}$ describes two alarm events, both related to the triples from our example LIG. The alarm events are reported to the operation and management platform, but administrators don't know the causal relationship given the huge list of alarms reporting similar alarms on different containers and services. By applying $R3$ on the whole KG, we learned triple 16, showing the causal relation between two alarm instances.

4.7 LEKG Performance

We applied LEKG to Huawei's 5G network logs exported from the OM platform. The data preparation consisted of two parts: (1) preparing a text corpus from raw logs for language representation learning and NER training; (2) preparing a log corpus for log extraction. The total size of 4 exported log files was about 1.2G compressed. The logs' formats vary as they were from multiple components of the 5G network, including network control, security, upgrade, operation, performance etc. We focused on Huawei's 5G unified network controller (UNC) logs, following the advice of Huawei engineers that most of logs from UNC are in a similar templates and record the service events related to network control. We also included service alarms from the OM platform. We parsed out log messages from each piece of log and created a text corpus for learning character-level representation. The NER annotation was also based on this text corpus. We also parsed out specific columns from logs based on templates to create semi-formatted log data for triple extraction. The semi-formatted log data was then aggregated by time slots according to designed test-to-failure test cases, for example, delete a service pod and recover it after a few minutes. The size of data are described in Table 4.1:

Table 4.1: Log Extraction Dataset

Original Log files (*.zip)	Cleaned UNC Text Corpus (*.gz)	Cleaned UNC logs (*.zip)
1.2G	21.9M	55.6M

The log extraction is a step by step process. We started by extracting high level concepts from Huawei's 5G network UNC manual. We used this initial KG as a BKG. It defines a set of concepts, relationsproperties, domain, range and property characteristics in OWL2, such as 'inverse of' and subsumption based on textual extraction and expert knowledge. We also designed 11 positive Horn rules and 5 negative rules based on expert knowledge. Then we

extracted triples from alarms to extend the BKG, which records events that are visible to system maintainers and developers. The triples extracted from alarms contain information about alarm parameters, such as pod ids, services ids and alarm causes. Then we extracted triples from logs and continued enriching the BKG.

The axiom numbers accumulated from different data sources are described in Table 4.2

Table 4.2: Log Extraction results

	UNC manual	Alarms	UNC logs (1%)
Total assertions	6434	45576	45784
Relational assertions	5410	39202	39401
Type assertions	1024	6374	6383
Relations	16	22	26
Entities	991	6322	6325
Concepts	30	-	- height

We applied template-based methods to extract background knowledge graph from the UNC manual. Then we used the cleaned UNC logs to extract triples and merge them into the BKG. Although the local to global reasoning approach reduces query space comparing to applying reasoning on the whole graph, the rule-based extraction still takes time with the Pellet reasoner. Inferring new triples from the same subset of cleaned UNC logs (about 10M) takes about 39 minutes on a Linux server with 3.60GHz Intel i7-6850K CPU and 64G memory, while it took 50 minutes on a MacBook with 2.3 GHz Quad-Core Intel Core i5 and 8G memory. The execution times didn't make a big difference, because the Pellet reasoner's main algorithm is NEXP2-TIME-complete. Each LIG reasoning with the Pellet reasoner takes 0.5 to 0.8 seconds on both computers. The Pellet reasoning process runs extremely long on the global KG, which is not applicable in real scenarios and it is not possible to compare the performance between reasoning on LIG and reasoning on Global KG. To further reduce the inference execution time, we implemented a query-based approach to infer the KG and learn new triples, which converts each Horn rule to a SPARQL query and searches the individual LIG directly. The SPARQL query doesn't involve any reasoning, hence some implicit connections are overlooked. But given that the LIG ontology is rather simple, we did not miss many new triples. The performance of two rule-based triple extraction implementation is described in Table 4.3. We name the constructed KG as *TREAT* (Tacit Relation Extraction and Transformation) for

Table 4.3: Log Extraction Performance

Size of cleaned UNC log subset (*.cvs)	17M
Number of axioms by reasoner	45784
Number of axioms by query	45738
Time with Pellet reasoner	5 hours
Time with query	20 minutess

reference in following chapters.

4.8 Complete Log KG with SICKLE system

With the extraction strategy described in the previous section, we finally construct TREAT dataset with 27,487 triples of network services, components and events. The schema of TREAT has 35 concepts, 18 object properties and a few object property axioms, such as symmetric object property, object property domain and range. We adopt the domain and range axioms as constraints, for instance, given $Domain(r) = A$, domain constraint demands that h must be an instance of A.

The literal embedding-based method in SICKLE encode entities with literal embedding. This is different from the NELL-995 and DBped-P datasets which used BERT as the text encoder in Chapter 3, the TREAT dataset uses a domain specific FastText [6] pre-trained language model as text encoder. Because the literal information from TREAT is very domain specific, the BERT pre-trained model doesn't work well on it. Moreover, the log corpus for constructing the TREAT dataset contains a lot of acronyms and sub-words, and usually does not follow a natural language grammar [71]. We then trained a character level language model by FastText from a log corpus related to the TREAT KG, and used it as text encoder.

We follow the same pipeline setting as in Chapter 3: we ran each triple producer in either a single-pass or 3 iterations. Also, we ran pipelines with combined triple producers in an iterative manner. We evaluated individual methods and combined pipelines in both schema-aware silver completeness ratios and schema-aware silver standard matrix.

We ran the combined pipeline with the TREAT dataset on a Linux server with a 3.60GHz Intel i7-6850K CPU and 64G memory. The execution time of a serialized combination was less than 1 hour in the first iteration, while it increased to 2 hours in 3 iterations as the number of triples increased.

The results are illustrated in both figures and tables:

1. Table 4.4 describes the results of schema-aware silver completeness ratios which are schema-correctness rate and coverage. We tested different types of KGC methods and their combinations in an iterative manner.
2. Table 4.5 describes the results of schema-aware silver standard LP, and Table 4.6 describes the results of TP. We tested the R-method, the E-methods and the L-methods on both LP and TP tasks based on the schema-aware silver standard.

6. https://fasttext.cc/

Our findings on the TREAT dataset are similar to DBpedia and NELL995 as discussed in Chapter 3, indicating that combining different types of triple producers can yield more schema-correct triples. However, when we apply SICKLE to the TREAT dataset, we uncover additional insights:

1. The schema-aware sampling strategy decreased the performance in LP and TP on the first iteration but, after three iterations, it improved performance in TP. There are a few possible reasons for the decreased performance on the TREAT dataset. Firstly, the schema-aware negative sampling strategy ignored the schema-correct negative samples; the bias it brings is amplified on a small dataset, which in turn affects the evaluation matrix of LP. Secondly, the TREAT literal encoder contains less context information than BERT, it doesn't work as well as it should when the training set is small. But when we do several rounds of training by injecting new schema-correct triples, it catches up and exceeds the E-methods in TP.
2. However, literals do not yield the anticipated benefits to the TREAT dataset in type prediction during the first iteration. A possible reason is that the text encoder for the TREAT dataset is very different from the encoder that was used for NELL-995 and DBped-P. The text encoder for TREAT literal was trained on the technical manual and the log messages via FastText, which was trying to catch sub-words and acronyms, but didn't involve much entity and class information. Two similar entity names may only be similar in their surface form, but distinguished in context, which increases the challenge of training with the literal encoder. This is another observation that literal qualities affect L-method's performance.

Table 4.4: Comparison of the $f_{Correctness}$ and $f_{Coverage}$ between individual and combined completion:

Iter.	M-method		E-method		L-method		R-method		Combined	
	Cov	Corr	Cov	Corr	Cov	Corr	Cov	Corr	Cov	Corr
1	0.01	1	2.19	0.87	2.54	0.79	2.43	0.70	5.62	0.87
2	-	-	3.97	0.92	3.65	0.72	5.87	0.98	**9.76**	0.85
3	-	-	5.01	0.94	4.72	0.67	8.78	0.98	-	-

4.9 Summary

In this Chapter, we introduced a system LEKG for automated knowledge graph construction from arbitrary log data. The proposed system extracts and learns triples from unstructured logs to construct a log KG based on a BKG. The key idea of this extraction approach is to utilize Horn rules from background knowledge to infer additional triples and validate new triples. From the angle of practice, we proposed a local to global strategy for triple inference, which reduced reasoning query space. Then, we demonstrated the knowledge extraction process and how it facilitates root cause analysis.

Table 4.5: TREAT datasets schema-aware silver standard evaluation in LP.

Model	Measure	E	L	E_{neg}	L_{neg}
TransE	Hits@1	**0.05**	0.03	0.04	0.03
	Hits@3	**0.10**	0.06	0.10	0.08
	Hits@10	**0.27**	0.12	0.24	0.23
	MRR	**0.11**	0.06	0.09	0.09
SimplE	Hits@1	**0.13**	0.03	0.08	0.02
	Hits@3	**0.25**	0.08	0.15	0.02
	Hits@10	**0.41**	0.19	0.27	0.02
	MRR	**0.22**	0.08	0.14	0.02
ComplEx	Hits@1	**0.11**	0.03	0.10	0.03
	Hits@3	**0.23**	0.09	0.18	0.07
	Hits@10	**0.40**	0.22	0.32	0.17
	MRR	**0.18**	0.09	0.16	0.07

Table 4.6: TREAT schema-aware silver standard evaluation in TP.

Model	Iteration	Measure	E	L	E_{neg}	L_{neg}
TransE	1	Pr	0.87	0.86	0.86	0.87
		Rec	0.85	0.82	0.90	0.85
		F1	0.86	0.84	0.89	0.86
	3	Pr	0.88	0.94	0.96	**0.95**
		Rec	0.83	0.91	0.93	**0.96**
		F1	0.86	0.92	0.94	**0.95**
SimplE	1	Pr	0.86	0.85	0.93	0.87
		Rec	0.90	0.84	0.90	0.86
		F1	0.89	0.84	0.92	0.87
	3	Pr	0.88	0.91	0.94	**0.94**
		Rec	0.86	0.93	0.93	**0.94**
		F1	0.87	0.92	0.94	**0.94**
ComplEx	1	Pr	0.88	0.86	0.88	0.88
		Rec	0.86	0.87	0.86	0.91
		F1	0.87	0.87	0.87	0.90
	3	Pr	0.89	0.93	0.93	**0.98**
		Rec	0.89	0.93	0.93	**0.95**
		F1	0.89	0.93	0.93	**0.97**

We applied the SICKLE system on this extracted log KG to produce new triples and evaluate the output with Schema-aware Silver Completeness Ratios and Schema-aware silver standard described in Chapter 3. Our finding are similar to those in Chapter 3, that combination pipeline and schema-aware sampling strategies can increase the schema-consistent ratio in the production of KGC pipelines. However, the L-method relies on the quality of the text encoder, which didn't perform better than the E-method in the TREAT dataset. This can be addressed in future work.

Chapter 5

Two Extensions of Schema-aware Combined KGC

In this chapter, we address the second research question through two strategies. These two strategies can be utilized as extensions of SICKLE, but they also have universal usage in improving the performance of link prediction tasks. In the first part, we implement a more refined method to combine multiple models. In the second part, we further explore solutions to the false negative problem with the assistance of LLM. This part of the work is an extension of the sampling strategy of the SICKLE system.
Q2: How to encourage the KGC models to produce more consistent triplets while ensuring accuracy in link prediction?

5.1 Probabilistic Ensemble of Multiple KGC models

5.1.1 Problem Statement

The SICKLE system described in Chapter 3 combined multiple KGC models and an ACC module to produce schema-correct triples. The underlying assumption of these approaches is that different KGC models have their own specific characteristics, such as modeling for a particular feature. Therefore, combining different models can compensate for their individual limitations and yield additional benefits. SICKLE's combination is based on a data sharing idea where multiple KGC models operate separately and benefit each other with their top-ranked schema-correct triples fed to iterative training. The new predictions are picked up based on individual model's scoring function. However, this combination strategy did not consider the trustworthy of the combined KGC models. When select new data to expand the KGs, there is a lack of comprehensive confidence measures from multiple models. The goal of the improvement is to combine multiple KGC models to expand KGs with high-quality triples based on the confidence from multiple KGC models.

There are some existing works [3, 81, 141, 171] that aggregate multiple KGC predictions based on models' prediction score. For instance, [141] combined the outcome of rule based link prediction and embedding based link prediction in a post-processing step, in which KGE scores are used as an additional information to change the position in the ranking of rule-based link predictions. It only combined two types of KGC methods by applying constant weights on two prediction scores. HybridFC [171] concatenates features from different KGC methods, and calculate final scores with a multi-layer perceptron model. These works combine models in different ways, and they achieve performance improvements in specific tasks. **However, it is unclear whether these improvements stem from the individual models themselves or from the specific combination methods used.** In other words, why does the combination method work?

To improve SICKLE with aggregated confidence score of link prediction, we develop a probabilistic ensemble of multiple KGC models. Moreover, we would like to explore whether a specific aggregation or fusion method makes distinguished link prediction performance. We aim for aggregation methods that can achieve higher quality in link prediction task while ensuring efficiency. In summary, our research in this section is to answer:

1. Why do we need combined KGC?
2. How to combine multiple KGC models in a post-processing manner and how to calculate a compound confidence score based on multiple model outputs?
3. What level of performance difference can different fusion methods bring?

The experimental analysis in this section is an extension of SICKLE system, where our conclusions will demonstrate the combined KGC is superior than single model in link prediction task. More important, we demonstrate a simple but efficient method to calculate confidence of link prediction with multiple KGC models.

5.1.2 Character of Embedding-based KGC and rule-based KGC

Two of the most common KGC methods are the KG embedding (E-method) and the logic rule learning (R-method).

Knowledge graph embedding approaches, such as [52, 53] (and many others), complete an input knowledge graph ABox by learning vector representations of existing entities and relations for predicting missing relations. The scoring function of the trained embedding model to score the new triples:

$$score = f_{embed}(\langle \vec{h}, \vec{r}, \vec{t} \rangle)$$

where f_{embed} is the scoring function of the embedding model.

Different KGE models capture different features and represent relations and entities by their unique structures and predict by various scoring functions. In Chapter 2, we listed the KGE models used in this thesis, along with their known characteristics and limitations.

Unlike teh KGE models, the logic rule learning is explainable and can provide insights for inference results. Using the rule body, the confidence of candidate rules is calculated by dividing number of supports with number of groundings with relation r in the KG.

To better understand the performance and characteristics of different KGC models, we conducted a set of experiments to compare the performance of individual models on different triple partitions. According to the analysis in [172], usually, specific models have better performance on the specific triple set. We test a list of KGC models on categorized test sets grouped by individual relation. We found that some KGC models have specific good performance on certain set of triples.

In this experiment, we focus on the rank based metrics for link prediction. Hence, we chose two widely used link prediction datasets: FB15K237 and WN18RR. We trained 4 typical KG embedding models, CP [138], ComplEx [95], TuckER [137], RotatE [53] on FB15K237 and WN18RR. Additionally, we included AnyBURL [163] which is a rule-based KGC model. We employed the ComplEx implementation and hyperparameters from [138], and for TuckER and RotatE, we used implementations and hyperparameters from PyKeen [173].

Each dataset is split into 3 subsets for training, validation, and testing. Furthermore, we categorized the test set triples based on their relations. Table 5.1 presents the breakdown of performance of each model with 10 randomly selected relations on FB15k237 dataset. Table 5.2 provides a performance breakdown for each relation on WN18RR.

Analyzing the results in Table 5.1 and Table 5.2, we observed that different models exhibited varying performances within the same group. As a result, our objective is to find a way to combine these diverse models to leverage their individual strengths, thereby enhancing the overall performance.

5.1.3 Combination of different KGC models

To confirm the superiority of the combination approaches than single method, we conduct a series of experiments to compare the performance of individual models and their combinations using post-processing techniques. Our post-processing approach involves assembling a set of KGC methods by combining the link prediction results from different models and re-rank candidates based on fused scores. We tested several fusion approaches, including simple average, weighted average, neural network, Platt Scaling and pick-the-best. We formalize the post-processing aggregation as below:

Let (h,r,t) be a test triple, for head or tail queries $(?,r,t)$ or $(h,r,?)$,
$v_h = \left[f(\vec{e_1},\vec{r},\vec{t}),...,f(\vec{e_k},\vec{r},\vec{t})\right] \in \mathbb{E}^k$, or $v_t = \left[f(\vec{h},\vec{r},\vec{e_1}),...,f(\vec{h},\vec{r},\vec{e_k})\right] \in \mathbb{E}^k$, be a vector of link prediction scores across all candidate entities in E. We have multiple link prediction models and each outputs corresponding candidate score vector $v = [v_h, v_t]$. We represent the

Table 5.1: FB15K237 Relation groups' performance in link prediction

Relations	num_train	num_test	ComplEx	CP	TuckER	RotatE	anyburl
/film/film/release_date_s./film/film_regional_release _date/film_release_region	15568	1447	**0.359**	0.357	0.359	0.291	0.319
/people/person/profession	13382	1311	**0.423**	0.391	0.394	0.400	0.355
/common/topic/webpage./common/webpage/category	4353	394	0.503	0.495	0.510	0.504	**0.519**
/location/location/contains	5830	328	0.392	0.365	**0.407**	0.314	0.396
/film/film/language	3125	314	0.490	0.467	0.496	0.470	**0.500**
/olympics/olympic_sport/athletes./olympics/ olympic_athlete_affiliation/country	2622	258	0.435	0.424	**0.437**	0.408	0.370
/education/educational_degree/people_with_this _degree./education/education/major_field_of_study	618	63	0.575	0.549	0.560	**0.623**	0.374
/travel/travel_destination/climate./travel/travel_ destination_monthly_climate/month	652	60	0.979	0.979	**0.986**	0.974	0.922
/people/person/spouse_s./people/marriage/ location_of_ceremony	307	35	0.098	0.098	0.092	0.108	**0.208**
/film/director/film	886	19	0.183	0.187	0.173	0.163	**0.337**

Table 5.2: WN18RR Relation groups' performance in link prediction

Relations	num_train	num_test	ComplEx	CP	TuckER	RotatE	anyburl
_derivationally_related_form	31865	1074	**0.961**	0.958	0.954	0.906	0.956
_hypernym	36873	1067	**0.182**	0.174	0.128	0.139	0.151
_member_meronym	7912	246	**0.280**	0.220	0.185	0.203	0.240
_has_part	5131	169	**0.241**	0.192	0.162	0.155	0.153
_instance_hypernym	3118	112	**0.426**	0.425	0.392	0.361	0.328
_synset_domain_topic_of	3328	110	**0.375**	0.341	0.305	0.321	0.360
_also_see	1396	56	0.632	**0.661**	0.639	0.545	0.643
_verb_group	1220	39	**0.975**	0.968	0.974	0.786	0.974
_member_of_domain_region	981	26	0.228	0.161	0.208	0.305	**0.394**
_member_of_domain_usage	673	22	**0.377**	0.253	0.293	0.302	0.342
_similar_to	86	3	1.000	1.000	1.000	1.000	1.000

prediction scores from multiple models as $\left[v^1,...,v^n\right]$, where n is the number of LP models. Our goal is to learn a function that transforms the score vectors $\left[v^1,...,v^n\right]$ into probability vector $prob$. The final prediction answer e to the query and corresponding confidence score $prob\,(e)$ are taken as the top ranked ones in E.

The input features for each aggregation method consist of prediction scores produced by each individual KGC model and evaluation performance on specific triple subsets. As suggested in [172], specific models tend to exhibit better performance on the particular subsets of triples. In their analysis, KGC models were evaluated on different categories within the test set. For example, the test set was categorized into 4 distinct groups based on relation mappings: 1-to-1, 1-to-n, n-to-1 and n-to-m. We experimented with three categorization strategies, group by relation, relation mapping, or entity degree.

Our fusion strategy takes into account two key factors: the prediction score of the triple on individual models and the performance metrics of the triple's respective group by individual models.

5.1.4 Fusion methods

Calibration

The scores from each KGC model are not necessarily on the same scale, and may not be directly interpreted as probabilities, as pointed out in [174]. To address this issue, some previous studies have employed an expit transform [175], which involves passing these scores through the Sigmoid function, to convert raw KGC scores into probabilities.

$$score\,(h,r,t) = f_{LP}\left(\vec{h},\vec{r},\vec{t}\right) \tag{5.1}$$

$$\sigma\,(score) = \frac{1}{1+exp\,(-score)} \tag{5.2}$$

However, later work [174] showed that probabilities obtained in this way cannot reflect probability, in other words, they are uncalibrated. Calibration is the technique to adjust the uncalibrated scores into calibrated probabilities. To calibrate a KGC model, calibration parameters are learned using the prediction scores of the KGC model on a validation set, and then these learned parameters are applied on test set. Therefore, calibration is a post-processing step that does not impact the trained KGC models.

Several off-the-shelf calibration techniques are available, such as Platt Scaling (or Logistic Calibration) [176], Isotonic Regression [177], Histogram Binning [178] and many others [174]. In [174], it reported Isotonic Regression performs better on large datasets (say, over 10 thousand), while scaling-based techniques like Platt Scaling are suitable for smaller datasets. For our study, we selected the well-known methods of Platt Scaling and Isotonic Regression to transform the expit-transformed KGC scores into probabilities within the range $[0,1]$.

We utilize calibration in two ways:

1. We input the scores from multiple KGC models as boxes of features and transform boxes into final probabilities.
2. We calibrate scores from individual models to probabilities and use the calibrated scores as input for fusion methods, for example simple average.

The first method directly produces fused probabilities using the following equation:

$$prob\,(h,r,t) = f_{calib}\left(\left[\sigma\left(score^1\right),...,\sigma\left(score^n\right)\right]\right) \tag{5.3}$$

The second method aims to transform the scores from different KGC models into similar scale, resulting in probabilities for each individual models. We retain the calibrated scores for downstream fusion methods.

Weighted Average

We group validation set either by relations or by relation mappings, and evaluate each group separately. We then employ performance metrics as weights to calculate average scores on the test set. Specifically, we use the corresponding group MRR scores as weights to average the scores from multiple models. The prediction scores of test set are calibrated before passing into the weighted average function:

$$prob\,(h,r,t) = \frac{\sum_{i=1}^{n}\left(prob^i \cdot w^i\right)}{\sum_{i=1}^{n} w^i} \tag{5.4}$$

Neural Network

We tested a linear classifier. The feature vector consists of calibrated scores from each KGC method. The validation data set is used as positive samples. The negative samples are selected among top ranked negative predictions based on closed world assumption (CWA). The classifier is trained to score the true triples higher than the negative triples.

Pick the Best

Similar to the weighted average method, we utilize performance metrics from groups (e.g., grouped by relations, relation mappings, or entity degrees) as indicators to apply weights to individual model scores. However, in contrast to the Weighted Average method, the Pick the Best method assigns a weight of 1 to the best-performing model and assigns a weight of 0 to the others.

$$prob\,(h,r,t) = \frac{\sum_{i=1}^{n}\left(prob^{i}\cdot w^{i}\right)}{\sum_{i=1}^{n} w^{i}} \tag{5.5}$$

$$w^{i} = \begin{cases} 1, \text{ if } w^{i} == Max(w) \\ 0, \text{ otherwise} \end{cases}$$

5.1.5 Experiment Setting

We test the above fusion methods with 5 KGC models: ComplEx [95], CP [138], TuckER [137], RotatE [53] and AnyBURL [163] on 2 datasets: FB15K237 and WN18RR.

We use the implementation of calibration techniques from NetCal [1] to calibrate raw scores from each KGC model. We used the linear classifier implementation from scikit-learn [2]. Our source code is publicly available at `https://github.com/sig4kg/KGFusion.git`. There are too many combinations with the 5 single KGC models. However, in order to save computational resources and keep the length of this document manageable, we will only test the following five combinations:

1. ComplEx + TuckER
2. ComplEx + AnyBURL
3. ComplEx + TuckER + RotatE
4. ComplEx + TuckER + RotatE + AnyBURL
5. ComplEx + CP + TuckER + RotatE + AnyBURL

We chose these combinations based on several considerations. Combination 1 and combination 3 are both combinations of KGE models. Combinations 2, 4, and 5 represent combinations of KGE and Rules. We aim to compare the performance of pure KGE combinations with combinations involving two different types of KGC models.

In combinations 4 and 5, we introduce rule-based KGC on top of combination 3, and we combine multiple models together. We aim to verify whether incorporating more combined models and rule-based KGC can provide additional advantages.

---

1. https://github.com/EFS-OpenSource/calibration-framework
2. https://scikit-learn.org/stable/index.html

In the basic SICKLE system, we combine embedding-based KGC and rule-based KGC to produce more schema-consistent triples, regarding the schema of the given KG. we have demonstrated that hybrid KGC pipeline can produce more schema-consistent triples than single method. However, we didn't pay much attention to the link prediction accuracy under the silver standard in the basic SICKLE system. In this experiment, we would like to know whether combinations of different type of KGC methods can improve the performance of link prediction. By testing and comparing these different combinations, we can gain a more comprehensive understanding of the impact and benefits of different types of KGC combinations on link prediction task.

5.1.6 Results and Analysis

Table 5.3 and Table 5.4 show the single models and the best scores of 5 combinations among 5 different combination methods. From these two tables, we can see that the combined methods generally achieve higher Hits@N and MRR than any single model. The KGE models are already good in aggregating different signals, while KGE+Rule can achieve even better scores. This result confirms that combining multiple different models can lead to performance improvements. However, we also want to further investigate the impact of different combination algorithms on the results. Table 5.5 and Table 5.6 shows the MRR results with 5 different combination methods. On WN18RR dataset, it is possible for a combination to achieve high scores on the other four fusion methods apart from Platt Scalling. Moreover, the differences in their scores are not significant. On FB15k237 dataset, the simple average fusion method performs well on most of model combinations. This indicates that a simple average or weighted average can yield good fusion results.

These insights provide a foundation for feasibility of our work in SICKLE: combining multiple different KGC models to complement each other. Moreover, the fusion approach does not need to be overly complex, as a simple fusion can yield better results than individual models.

5.1.7 Summary

We demonstrate that each KGC approach has limitations and that no single method can handle all of the possible scenarios in KG. Different KGC models employ distinct techniques, architectures, or assumptions. By combining models that have diverse approaches, we can capture complementary information and perspectives on the knowledge graph. This diversity can help uncover latent patterns and improve the overall performance of link predictions. We train several individual KGC models and calibrate each model to get properly scaled probabilities. Then, we test a various of fusion methods, including average, calibration, voting, or a new learner, to re-rank candidates. Our findings are as follows:

1. Ensembling KGE models improves predictive performance, and the ensemble of KGE + rule-based models performs even better.

Table 5.3: The LP results on single model and combined methods on WN18RR dataset.

Models	Head				Tail				Both			
	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR
RotatE	0.375	0.453	0.521	0.428	0.429	0.530	0.617	0.496	0.402	0.492	0.569	0.462
TuckER	0.430	0.462	0.507	0.455	0.477	0.511	0.551	0.502	0.453	0.486	0.529	0.479
ComplEx	**0.448**	**0.511**	**0.586**	**0.495**	**0.487**	**0.560**	**0.643**	**0.539**	**0.468**	**0.535**	**0.614**	**0.517**
CP	0.444	0.495	0.562	0.483	0.483	0.540	0.598	0.525	0.464	0.517	0.580	0.504
AnyBURL	0.442	0.496	0.569	0.485	0.503	0.560	0.616	0.544	0.472	0.528	0.592	0.514
CPL+TR	0.459	0.515	0.592	0.502	0.480	0.557	0.639	0.534	0.470	0.536	0.615	0.518
CPL+AB	0.462	0.519	0.594	0.505	0.506	0.574	0.645	0.553	0.484	0.547	0.619	0.529
CPL+TR+RT	0.459	0.516	0.595	0.503	0.479	0.557	0.641	0.534	0.469	0.537	0.618	0.518
CPL+TR+RT+AB	**0.465**	**0.520**	**0.588**	**0.507**	**0.516**	**0.575**	**0.645**	**0.558**	**0.491**	**0.547**	**0.616**	**0.532**
CPL+CP+TR+RT+AB	0.461	0.511	0.590	0.502	0.513	0.581	0.646	0.560	0.487	0.546	0.618	0.531

Table 5.4: The LP results on single model and combined methods on FB15k237 dataset.

Models	Head				Tail				Both			
	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR
RotatE	0.153	0.264	0.421	0.241	0.331	0.483	0.640	0.435	0.242	0.373	0.531	0.338
TuckER	0.168	0.287	0.443	0.260	0.353	0.496	0.642	0.451	0.261	0.391	0.542	0.355
ComplEx	**0.176**	**0.296**	**0.457**	**0.268**	**0.354**	**0.498**	**0.651**	**0.453**	**0.265**	**0.396**	**0.554**	**0.361**
CP	0.163	0.279	0.435	0.252	0.349	0.489	0.637	0.446	0.256	0.384	0.536	0.349
AnyBURL	0.150	0.254	0.398	0.233	0.327	0.456	0.605	0.420	0.238	0.355	0.501	0.327
CPL+TR	0.183	0.300	0.466	0.275	0.365	0.506	0.658	0.462	0.274	0.403	0.562	0.369
CPL+AB	0.189	0.307	0.464	0.279	0.364	0.506	0.656	0.462	0.276	0.406	0.560	0.370
CPL+TR+RT	0.183	0.300	0.466	0.275	0.364	0.506	0.658	0.462	0.274	0.403	0.562	0.369
CPL+TR+RT+AB	**0.190**	**0.311**	**0.468**	**0.281**	**0.371**	**0.514**	**0.664**	**0.469**	**0.280**	**0.413**	**0.566**	**0.375**
CPL+CP+TR+RT+AB	0.189	0.309	0.466	0.280	0.370	0.511	0.663	0.468	0.280	0.410	0.565	0.374

Models	Average	WAverage	Platt Scalling	Neural Net	Pick the Best
CPL+TR	0.513	**0.518**	0.460	0.500	**0.518**
CPL+AB	**0.529**	0.518	0.459	0.487	0.520
CPL+TR+RT	0.517	**0.518**	0.478	0.509	**0.518**
CPL+TR+RT+AB	0.524	0.527	0.497	**0.532**	0.519
CPL+CP+TR+RT+AB	0.527	**0.531**	0.523	0.511	0.519

Table 5.5: The LP MRR results of combined methods on WN18RR dataset.

Models	Average	WAverage	Platt Scalling	Neural Net	Pick the Best
CPL+TR	0.368	0.366	0.367	**0.369**	0.366
CPL+AB	**0.370**	0.368	**0.370**	0.358	0.368
CPL+TR+RT	0.368	0.366	0.316	**0.369**	0.366
CPL+TR+RT+AB	**0.375**	0.369	0.359	0.364	0.370
CPL+CP+TR+RT+AB	**0.374**	0.370	0.365	0.372	0.370

Table 5.6: The LP MRR results of combined methods on FB15k237 dataset.

2. Almost all of these assembly methods, such as averaging, calibration, voting, or neural network, can improve performance. Even the simple averaging leads to significant improvements. Therefore, the specific assembly method used plays a less critical role in the combination's performance, while the combination itself is the key driver.

These insights have given us more confidence in combining multiple KGC models to accomplish the same LP task.

5.2 From Schema-aware to Informative Negative Sampling

5.2.1 Problem Statement

In Knowledge Graph Embedding (KGE), the goal is to represent entities and relations in a vectorial space, enabling efficient knowledge representation, reasoning, and applications such as link prediction, entity recognition, question answering and recommendation systems. During KGE training, a model learns to differentiate between positive triples and negative triples. Knowledge graphs (KGs) are often sparse, meaning that only a fraction of all possible triples are present in the graph. Most of existing KGs only store positive triples which are believed to be true. Negative samples are artificial triples created by replacing entities or relationships in existing triples, used to train models to distinguish true triples from false ones. Instead of using all possible negative examples, which could be computationally infeasible, negative sampling strategies allows the model to focus on a subset of negative samples. This makes the training process more efficient and scalable, especially for large KGs [166, 179].

Numerous strategies exist for generating negative samples in KGE. Classic uniform random sampling [52] randomly selects entities to replace either the head or tail of existing triple to create a negative sample. Other sampling strategies, such as Bernoulli sampling [74], importance sampling [75], adversarial negative sampling [76–78] and dual-encoder sampling [75] make effort to generate more challenging negative samples that are difficult for the primary KGE model to distinguish. However, the majority of these negative sampling strategies are based on the Closed World Assumption (CWA), which presupposes that the absence of a fact (a triple) in the KG implies the falseness or nonexistence of that fact. This can lead to the false negative problem. For instance, if the KG contains the triple $(John, likes, IceCream)$, under the CWA, the negative sample $(John, likes, Pizza)$ would be unequivocally considered as false, suggesting that John does not like pizza. In contrast, the open-world assumption (OWA) treats missing triples as unknown or unobserved.

Schema-aware sampling, shown in our previous studies in Chapter 3, leverages the KG's schema (TBox) to identify negative triples that are *inconsistent* with the existing KG, thus mitigating the problem of false negatives. This approach assumes the existing KG is up-to-date and asserts that unknown facts should follow the constraints of the existing KG; otherwise, they are considered false. While schema-aware negative sampling has addressed the false negative problem to some degree, it may sometimes include inconsistent triples and overlook statements that, although consistent, are incorrect.

To address this limitation, we revisit the schema-aware sampling strategy proposed in our work in Chapter 3 and introduce an open-world schema-aware sampling strategy. This novel approach aims to not only include the schema-inconsistent negatives but also negative samples that are both informative and likely to be false under the OWA. Specifically, in our negative sampling strategy, we consider three different sets of candidates:

Schema-inconsistent triples We make usage of our previous work described in Chapter 3, especially, using its approximate consistency checking (ACC) method to identify schema-inconsistent triples from high ranked baseline knowledge graph completion (KGC) predictions. These inconsistent triples serve as negative examples.

Schema-consistent yet negative triples Recent research [180] has demonstrated that large language models like ChatGPT can be invaluable tools in document ranking tasks, often delivering results that are competitive or even superior to those achieved by supervised methods on widely-recognized information retrieval benchmarks. To re-rank the schema-consistent triples, we model this task as a query-answering task. We leverage ChatGPT to re-rank a list of candidates based on their relevance to the query triple missing head or tail. Less relevant candidates are highly likely to be assigned a negative status. Consequently, we propose a prompt to instruct ChatGPT to directly provide the permutations of a group of candidate entities.

Informative negative triples from peer-based statistical inferences The informative negative statements that are produced by peer-based statistical inferences [157, 159], derives noteworthy negative statements by combining information from highly related entities, namely peers, with supervised calibration of ranking heuristics. The underlying idea of this method is that similar entities can suggest expectations for relevant statements about a given entity. For instance, many peers of Stephen Hawking, notably other physicists, have won the Nobel Prize in Physics. From this, we can infer that it is plausible to expect that he also won this prize, rendering the fact that he did not win it an especially informative statement.

Our objective in this section is to generate negative samples more likely to be false under open-world assumption, and feed them to KGE training.

The schema-aware negative sampling strategy selects negative samples that cause inconsistency when being added to the KG [181]. Our open-world schema-aware negative sampling strategy is an upgrade to the schema-aware negative sampling strategy. Specifically, we generate negative samples that are consistent with the existing KG, but likely to be incorrect under the OWA. Furthermore, we favour the selection of the most informative negative samples. In [155, 159, 182], informative negative statement means that the head and tail phrases must be thematically or topically consistent and noteworthy, for instance, $(TomCruise, won, Oscaraward)$ is an informative negative statement [157, 159] as opposed to winning NBA Best player award for example.

We formalize language models and open-world schema-aware negative sampling strategy to two forms of negative statements:

$$N_{incon} = \{r(h,t) \mid \mathscr{G} \models r(h,t) \sqsubseteq \bot, r(h,t) \notin \mathscr{G}\} \tag{5.6}$$

$$N_{con} = \{r(h,t) \mid \mathscr{G} \models r(h,t), r(h,t) \notin \mathscr{G}, f_{LM}(r(h,t)) < \lambda\} \tag{5.7}$$

We addressed N_{incon} in our previous work [181] by employing an iterative KGC pipeline that uses ACC to identify the inconsistency subsets from an expanded KG. For the generation of N_{con}, we employ ChatGPT to rank the predicted statements consistent with the input KG. ChatGPT, as a language model, can use its contextual understanding of language to infer whether a triple is plausible or contradictory based on the existing knowledge. KGs can have ambiguous information, and ChatGPT may not always provide definitive answers in such cases. It might generate responses based on probabilities and not necessarily based on absolute correctness. For each triple query missing head or tail and a list of candidate entities from baseline KGC ranking, we generate prompts by converting it to a sentence. It prompts ChatGPT to rank the candidates with scores to show likelihood. Then, we select the most unlike candidates as negative samples.

Additionally, we incorporate pre-generated informative negative statements, namely Wikinegata [157, 159] (negated statements about Wikidata [3] entities), denoted as N_{psi}, which are produced using peer-based statistical inferences. As triples in N_{psi} are also consistent, we merged it to N_{con}. The final set of negative samples is drawn from the union of N_{incon}, N_{con}.

5.2.2 Structure of the Negative Sampling Strategy

With all three sets of negative statements, we create a negative sample pool to be fed in KGE training. Figure 5.1 illustrates the structure of the negative sampling strategy.

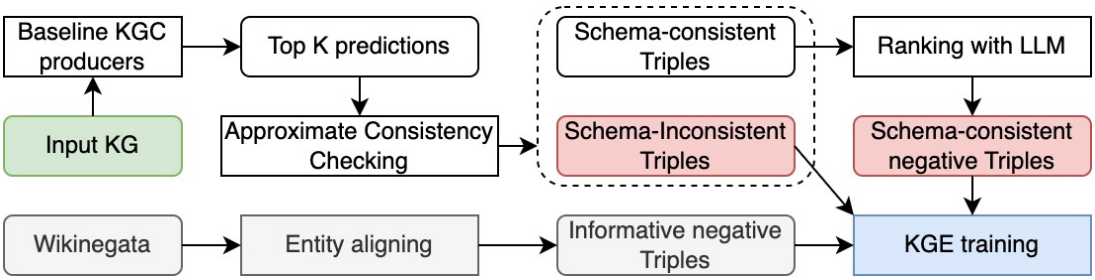

Figure 5.1: The Negative Sampling Strategy

5.2.3 Negative Sampling with Approximate Consistency Checking

The previous negative sampling strategies under CWA, such as uniform random sampling, draw from large sample spaces of negative samples [80, 157]. We attempt to reduce the sample space by selecting the top-ranked predictions in a baseline KGC pipeline, such that we increase the likelihood of selecting negatives that are relevant to the task at hand. The baseline KGC system combines a rule-based KGC model, namely AnyBURL [163] and a well known KGE model ComplEx [95] to produce negative candidates. We first remove the known positive triples from the predictions. We then apply ACC to identify inconsistent subsets. The ACC approach allows a detection of inconsistencies in a KG by using the so-called *inconsistency justification patterns* (IJPs) [70, 181] shown in Appendix A. The idea is to identify possible IJPS from the TBox, and scan the ABox to find subsets that match the IJPS. Here, we explain the ACC algorithm with an example:

Example 8. *Given these axioms:*
$Range(has\_parent) = Person$
$Range(works\_for) = Organisation$
$DisjointWith(Person, Organisation)$
$has\_parent(Anna, Lewis)$

3. https://www.wikidata.org

$works\_for(Mary, Lewis)$
And a designed IJP:
TBox consistency checking pattern: $\exists r1^- \sqcap \exists r2^- \sqsubseteq \bot$*;*
ABox violating triples: $\langle e2, r1, e1\rangle, \langle e3, r2, e1\rangle$

We get inferred axioms based on the designed IJP:
1. $\exists has\_parent^- \sqsubseteq Person$
2. $\exists works\_for^- \sqsubseteq Organisation$
3. $\exists has\_parent^- \sqcap \exists works\_for^- \sqsubseteq \bot$

The expression $\exists has\_parent^- \sqcap \exists works\_for^- \sqsubseteq \bot$ is able to identify the schema-inconsistent subset in Example 8: any subset in the form of $(works\_for(x, e), has\_parent(y, e))$ is schema-inconsistent subset. Then we justify the axioms in Example 8 is a schema-inconsistent subset. The detail of our ACC algorithm and a list of designed IJPs are described in [181]. We regard triples as either schema-consistent, schema-inconsistent, or schema-unknown in terms of their compliance with the existing KG [3, 181]. In this context, schema-inconsistent triples are true negative samples.

5.2.4 Informative Negative Sampling with Large Language Model

We leverage large language model such as ChatGPT3.5-tubo [4] and ChatGPT4 [5] to rank the schema-consistent or schema-unknown candidates. Example 9 shows a query of missing tail entity with uncertain answer. A population place refers to an area where members of an ethnic group reside. English Americans are widely distributed across many states; however, there are limited populated places associated with them in DBpedia. For instance, DBpedia does not list cities in New England as populated places of English Americans. We select those scores less than a threshold as negative samples.

Example 9. *Rank candidates among [New England, Hawaii, Arizona, New York metropolitan area, Vietnam, Uruguay, Seattle metropolitan area, Chicago metropolitan area, Guatemala] based on the relevance to the query* $\langle EnglishAmericans, populationplace, ?\rangle$*, with a score range of* $[0-1]$ *where 1 the true.*
Answer:
New England - 1; New York metropolitan area - 0.8;
Chicago metropolitan area - 0.7; Seattle metropolitan area - 0.6;
Hawaii - 0.5; Arizona - 0.4; Uruguay - 0.2 Vietnam - 0.1;
Guatemala - 0.1

4. OpenAI. 2022
5. OpenAI. 2023

5.2.5 Informative Negative Sampling with peer-based statistical inferences

The informative negative statements produced by peer-based statistical inference [157, 159], derive noteworthy negative statements by combining information from highly related entities, namely peers, with supervised calibration of ranking heuristics. For context awareness, we provide a concise summary of [157, 159]. Given an entity e, this approach starts by collecting the peer group of e. A peer group of e are related entities of e and their statements. These groups are obtained with various methods, such as entities sharing same class, entities sharing same property, entities having similar KGE, and so on. Next, for each peer group, all statements and properties that these peers have are retrieved, and ranked by their relative frequency. The candidate negative statements are created based on the peer's most frequent statements. Often, the candidate negative statements set is large, thus, it uses several factors to rank the candidates, such as relative frequency, popularity of the object entity (e.g. Wikipedia page views), and entity embedding similarity [159]. We reuse the pre-generated negative statements, namely Wikinegata [159].

5.2.6 Experimental Setting

Our notion of schema-aware in this work relies on 3 criteria: 1. high enough number of triples for KGE training; 2. a schema that has a rigid hierarchy of concepts and rich object properties; and 3. has textual literal to help generate prompts for ChatGPT. Hence, we performed experiments with two KGs: DBped-P [3] and DB15K [183]. We didn't include the TREAT dataset because it is more of a domain-specific KG, and its textual literals come from system logs that do not follow general grammar or vocabulary. Therefore, the TREAT dataset cannot meet the third criterion. To reuse the ACC algorithm in [181] and generate prompts for ChatGPT, we queried the official DBpedia SPARQL endpoint [6] to get type assertions and text literals. And we reduced the original DBpedia schema to a subset that only contains assertions, constraints and definitions that related to the two datasets. The details of two datasets are described in Table 5.7. A traditional evaluation strategy for KGC is to use a subset of the given

Table 5.7: Dataset Statistics

Dataset	Entities	Relations	rel assertions	Type assertions
DBped-P	157,879	188	352,754	932,200
DB15K	12,781	238	89,197	80,441

KG as a test set, often referred to as silver standard evaluation. A problem with the silver standard approach is that the KG itself might not be complete, thus could potentially produce false negative results. As for gold standard evaluations, the result quality is usually measured in recall, precision, and F-measure [1]. The silver standard evaluation is usually applied to

6. https://dbpedia.org/sparql

measure the performance of KGC approaches on how well a given triple is replicated by a KGC method, with rank based evaluation metrics Hits@n and Mean Reciprocal Rank (MRR) calculated against the test set [1]. Under the schema-aware KGC paradigm, those triples that do not exist in the initial KG but can be derived through deductive logic reasoning are also considered true triples [181]. In this paper, we use the *Schema-aware Silver Standard* [181]. Instead of splitting a subset of a given KG as a test set, our schema-aware silver standard is tested on a union of the KG's materialised triple set and the test set split from a given KG. We removed any inconsistency subsets from the original datasets before run materialisation with OWL 2 reasoners [92, 131, 135]. The KGE training is carried on with the consistent subset of the original KG, and we calculate Hits@n and MRR against the expanded test set.

We reused the implementation of basic KGE models from BLP [87] for its easy to extend design. The BLP utilizes a margin-based loss function and a vector dimension of 128. The basic KGE model implementation may not follow the state-of-the-art techniques recommended in [184] to achieve the best link prediction performance. However, it is sufficient to demonstrate the performance difference between negative strategies.

5.2.7 Result Analysis

Table 5.8 and Table 5.9 show the results of our negative sampling strategies on three well-known KGE models. We observed that the performance of the schema-aware negative sampling strategy with N_{incon} is higher than the performance of KGEs with uniform random sampling and Bernoulli sampling. This indicates that N_{incon} effectively improves the quality of negative sampling.

When we have full negative sampling pool with N_{incon}, N_{con}, we set higher priority on negative samples in N_{con} over N_{incon}, to make sure N_{con} are selected. Generally, the strategy with N_{con} had a slight advantage over single N_{incon}. Why did the N_{con} not bring a significant improvement over N_{incon}? The reason could be that N_{incon} introduced much more negative samples compared to N_{con}. When we re-rank N_{con} with ChatGPT API, in order to make the API response quickly, we limited each query to a maximum of 10 candidates. And the final selection is at about 2 candidates per query in average. The number of informative negative triples relies on how many entities and relations we can map between our experimental datasets and Wikinegata [159].

5.2.8 Summary

Negative sampling is a strategy to improve KGE training by allowing models to learn from non-existent (negative) triples, addressing data sparsity. Various strategies for generating negative samples exist, but many operate under the Closed World Assumption (CWA), considering absent facts as false. This can lead to false negatives. In contrast, the Open World Assumption (OWA) treats missing triples as unknown. We assume the exiting KG are up-to-date, and argue that unknown triples should follow the schema of existing KG, otherwise they are false. To address the false negative problem in CWA, we introduces an open-world schema-aware sampling strategy, generating negative samples while considering schema consistency and peer-based statistical inferences. Experimental results show superior performance in schema-aware link prediction tasks compared to baseline random sampling.

Table 5.8: Schema-aware silver standard link prediction result on DB15K dataset.

Methods	TransE		SimplE		ComplEx	
	MRR	Hits@1,3,10	MRR	Hits@1,3,10	MRR	Hits@1,3,10
Uniform Random	0.086	0.035/0.104/0.189	0.153	0.095/0.171/0.272	0.140	0.085/0.156/0.253
Bernoulli	0.144	0.094/0.162/0.245	0.148	0.095/0.167/0.256	0.151	0.098/0.167/0.258
N_{incon}	0.171	0.116/0.190/0.282	**0.177**	**0.124/0.193/0.289**	0.171	0.112/0.188/0.289
$N_{incon} + N_{con}$	**0.179**	**0.127/0.195/0.285**	0.174	0.121/0.190/0.284	**0.172**	**0.113/0.188/0.295**

Table 5.9: Schema-aware silver standard link prediction result on DBped-P dataset.

Methods	TransE		SimplE		ComplEx	
	MRR	Hits@1,3,10	MRR	Hits@1,3,10	MRR	Hits@1,3,10
Uniform Random	0.175	0.110/0.200/0.299	0.153	0.089/0.178/0.277	0.171	0.109/0.199/0.290
Bernoulli	0.189	0.118/0.215/0.325	0.165	0.106/0.185/0.283	0.170	0.109/0.190/0.289
N_{incon}	0.201	0.125/0.235/0.346	0.174	0.122/0.191/0.281	0.195	0.120/0.230/0.341
$N_{incon} + N_{con}$	**0.204**	**0.128/0.238/0.346**	**0.194**	**0.118/0.231/0.341**	**0.200**	**0.124/0.235/0.344**

Chapter 6

Conclusion

6.1 Summary of Findings

When we construct an initial KG of a certain scale, it is hard to say it's complete. Current research in Knowledge Graph Completion often emphasizes the growth aspect, focusing primarily on achieving completeness within Knowledge Graphs. However, there is a noticeable lack of research examining the consistency of the facts contained within Knowledge Graphs. A significant portion of existing approaches in Knowledge Graph Completion neglects the role of a Knowledge Graph's schema in ensuring the consistency of its triples. To tackle this problem, we utilize the schema of a Knowledge Graph to identify schema-inconsistent triples and subsequently enrich the Knowledge Graph by incorporating schema-consistent triples.

Our research centers around three key research questions: 1. how can we increase the number of consistent triples produced by a Knowledge Graph Completion approach with regards to a schema of a Knowledge Graph? 2. how can we encourage models to produce more consistent triplets while maintaining accuracy in link prediction? 3. in a real 5G network log system, given multiple complex data sources, how to make the a domain specific KG an evolving process by combining data-driven methods and knowledge-based methods, with regard to the schema of KG?

Regarding to the first research question, we encountered limitations of the previous work:

1. Approaches like [64, 65, 79, 142–147] focused on learning joint models that inject logic into KGEs to obtain more predictive entity and relation embeddings. However, they only address a limited range of ontological information and do not use the schema to decide whether a new triple is consistent or not.
2. Some methods [65, 79, 80] applied logic rules in their KGE sampling strategies to avoid the false negative problem and encourage KGE model to produce more consistent triples. Nevertheless, none of these methods guarantee that an expanded Knowledge Graph is consistent with the ontological schema of the original Knowledge Graph.

3. SIC [3] challenged the silver standard method, by proposing the notion of schema-correctness. SIC used an approximate consistency checking (ACC) method [70] with its triple producers to detect the inconsistencies that either already exist in the knowledge graph, or that are introduced by triple producers during an iterative process. SIC [3] only include schema-correct triples in its training set, while the inconsistent triples were abandoned. We argue that these inconsistent triples can serve as valuable negative samples during KGE training.
4. [65, 79] have focused on applying logic rules in their KGE sampling strategies to avoid the false negative problem and force KGE model to produce more consistent triples. But it only uses a small set of inconsistent triples in the optimal objective function for consideration of efficiency and scalability.
5. [80] proposed a method that leverages schema to dynamically generate inconsistent triples as negative examples in its training procedure. However, it requires training the model once in advance to generate adversarial negative samples. Its consistency checking strategy assumes the given ontologies are in DL-Lite. It considered the syntax and semantics that can be directly translated from OWL 2 to $DL-Lite^{S\sqcup}$, however it is not clear how it handles other syntax in OWL 2.

To address these limitations, we adopted several strategies in chapter 3:

1. We develope an approximate consistency checking service to identify inconsistent subsets in the produced triples. This ensures that only schema-consistent triples are included in an expanded KG.
2. We include a literal-embedding-based KGC method in our combined pipeline. Most real-world KGs evolve quickly with new entities and new triples being discovered over time. Recent years have witnessed increasing interest in learning KG representations with extra literal information [83–88]. The literal-embedding methods (L-method), such as DKRL [84], SSP [85], Conmask [83], BLP [87] and many others, have the potential of linking novel entities into existing KG, which referred as inductive LP [89]. Our goal in integrating the L-methods is not just to use it to handle new nodes, but we argue that combining literals and the per-trained language models can improve schema-correctness in KGC pipeline compared to pure KGE method. Our experimental results confirmed this hypothesis.
3. We develop a schema-aware sampling strategy to force the KGC pipeline produce more schema-consistent triples. Also, this sampling strategy targets the false negative problem that are not addressed in SIC [3] and other KGC works [52, 65, 74–80]. By exploiting logical consistency in the sampling strategy, we would like to encourage the models to predict more high-ranked consistent triples.

4. SIC [3] only include schema-correct triples in its training set, while the inconsistent triples were abandoned. We develop a schema-aware negative strategy which make use of the schema-inconsistent triples as negative samples during iterative KG embedding training procedure.
5. We implement an approximated consistency checking service taking into account both efficiency and rich expressive power by approximate TBox from OWL 2 to DL-Lite. The approximated consistency checking service is scalable and efficient enough to be integrated in a KGE model training procedure.
6. We integrate different types of KGC models and an approximated consistency checking service in a pipeline. The pipeline can be executed in series or in parallel in iterations. In iterations, each KGC method benefit each other by data augmentation that schema-consisitent triples are positive samples and schema-inconsistent triples are negative samples. We observed that combined pipelines are more productive than single method in producing schema-consistent triples. Moreover, we use the schema-inconsistent triples identified in each iteration as negative sample, so the schema-aware negative sampling strategy and KGE training become a convenient and natural combination.

With all these strategies, we developed a unified system called *SICKLE* for schema-aware iterative combined KGC. We experimentally tested SICKLE on both community benchmarks and an industrial dataset. Our main findings are highlighted as:

1. Including literal-embedding-based methods and pre-trained language models in combined KGC pipelines can encode additional schema-related features from literals while learning KGE, and these additional features are transferable to the downstream type prediction task.
2. By combining multiple KGC methods and exploit an approximated consistency checking service in models' training procedure, we produce more schema-consistent triples than single method.
3. The parallel KGC pipelines are significantly more efficient in producing schema-consistent triples than single model or series pipeline.

For the second question, we observed that there are three limitations in existing work:

1. [3, 81, 141, 171] aggregate multiple KGC predictions based on models' prediction score. These works combine models in different ways, and they achieve performance improvements in specific tasks. However, it is unclear whether these improvements stem from the individual models themselves or from the specific combination methods used.
2. [3] combines multiple KGC for producing schema-consistent triples. But, it ignores the confidence from individual models, and there is no trustworthy learned from its multiple models.

3. [65, 80] employ semantic reasoning to address false negative problem, so that their models tend to predict more consistent triples and also have positive impact on link prediction accuracy. While these work have addressed the false negative problem to some degree, it may sometimes include inconsistent triples and overlook statements that, although consistent, are incorrect.

To overcome the limitation of the existing work in the third research question, we improve the SICKLE system through two separate research effort. We described these strategies in Chapter 5.

1. In order to validate the superiority of the combination methods, we implement and test a series of fusion methods based on calibrated scores from different KGC models. We found that even the simplest averaging or merging of the probabilistic scores from multiple KGC models could achieve better link prediction performance than a single KGC model with 3% increment in Hits@1, and 1.5% in MRR on WN18RR dataset and FB15K237 dataset on average.
2. We introduce an open-world schema-aware sampling strategy, generating negative samples while considering schema consistency and informativeness. Experimental results show superior performance in schema-aware link prediction tasks compared to baselines. On average, there was approximately a 10% increase in MRR on both DB15K dataset and DBped-P dataset."

Our main findings are:

- Ensembling KGE models improves predictive performance, and the ensemble of KGE + rule-based models performs even better.
- Almost all of these assembly methods, such as averaging, calibration, voting, or neural network, can improve performance. Even the simple averaging leads to significant improvements. Therefore, the specific assembly method used plays a less critical role in the combination's performance, while the combination itself is the key driver.
- Expanding our sampling strategy from CWA to OWA with the help of schema, high-quality negative samples effectively enhances the performance of link prediction.

For the third question, we observed that a few log extraction work has been conducted in recent year. We summarize the limitations as follows:

1. Typical log analysis tasks begin with template-based detection to extract useful information, such as entities and properties, from logs [109]. Subsequently, data-mining algorithms are employed to analyze or summarize patterns from events. However, these approaches do not formalize events and their connections in formal knowledge representation, nor do they consider other perspectives on data.

2. Many existing log extraction works focus on template parsing within a segment of system logs. Due to the accumulative nature of log data, in the real-world scenarios, the knowledge grows as logs come on stream. As a result, reasoning [168, 169] covering a whole KG becomes increasingly resource-intensive over time. It is essential to keep the reasoning within a manageable query space.
3. Some existing log analysis approaches [102] only check if a reference entity exists, yet checking logical or semantic consistencies of the extracted knowledge has not been well addressed.

We address these limitations in Chapter 4. We have developed a system for extracting knowledge from logs to facilitate knowledge representation and reasoning. Our strategies include:

1. We combine a variety of NLP methods and logic rule-based extraction techniques to extract and infer knowledge from multiple data sources. Additionally, we have implemented constraint-based triple validation on new triples to ensure that they do not conflict with established rules.
2. We implement a local-to-global strategy for triple inference in the log extraction process, effectively reducing the reasoning query space.
3. We consider the extraction process as a triple producer and utilize the SICKLE system on the extracted log KG. The SICKLE system completes the log KG with schema-consistent triples.

Our findings of this work are summarised as:

- In scenarios with well-defined background knowledge, integrating rule-based inference with knowledge acquisition proves to be an effective approach for constructing a knowledge base. This method offers direct and efficient results in the scenario with complex data sources. Additionally, leveraging rule-based knowledge inference on this log KG can addressing root cause analysis to some degree.
- In scenarios with well-defined background knowledge and a continuous influx of vast amounts of data, partitioning the data sources and applying a local-to-global reasoning approach can effectively address the issue of excessively large reasoning spaces.
- Combination KGC pipeline and schema-aware sampling strategies can increase the schema-consistent ratio in the production of KGC pipelines in this domain specific KG. However, the literal embedding-based method relies on the quality of the text encoder, which didn't perform better than the pure embedding-based method in the TREAT dataset. This should be addressed in future work.

6.2 Limitation and Future Work

6.2.1 Temporal Knowledge Graph Completion

It might be interesting to explore how our proposed framework can be applied in dynamic settings, such as reasoning and learning for stream Knowledge Graphs [185] and temporal Knowledge Graph completion [186–188], where each fact is additionally associated with a timestamp. This including temporal representation, temporal reasoning and updating.

Our current log extraction system described in Chapter 4, lacks consideration for timestamps, yet these timestamps play a crucial role in understanding the temporal dynamics of a running system. In temporal knowledge graphs, facts transform into quadruples with the inclusion of timestamps, enabling a more nuanced understanding of temporal dynamics. In our future work on log extraction, it becomes imperative to represent temporal aspects in a structured and efficient manner. This entails attention to ontology design and data modeling, ensuring that temporal information is integrated into the downstream tasks, such as link prediction and root cause analysis.

Our SICKLE system described in Chapter 3 operates by producing schema-correct triples iteratively. This iterative approach is particularly well-suited for dynamic and temporal settings, where the data continuously evolves over time. However, leveraging this iterative KGC on temporal KGs requires additional attention to KG representation and reasoning algorithms. Temporal reasoning expects understanding the temporal dynamics of knowledge graphs with the provided timestamps of facts.

There are some existing works in this research area. The timestamp-included Tensor Decomposition, such as [189, 190], adopts the Canonical Polyadic Decomposition [191] by regarding knowledge graph as a 4-way tensor and expresses interactions among entities, relations, and timestamps. Timestamp-based Transformation models, such as [192–194], consider timestamps as a transformation to learn entity and/or relation representations. Works in [195, 196], learn from Knowledge Graph Snapshots corresponding to specific times. In this way, the knowledge graph becomes a temporal evolving sub-graph with varying relation connections. These works focus on modeling the temporal dynamics of KGs, but do not pay much attention to preserving data consistency. Furthermore, temporal knowledge graphs may evolve over time due to updates, additions, or deletions of entities and relations. Managing different versions of the knowledge graph and tracking changes while preserving data consistency and integrity is a significant challenge [197].

Addressing the challenge of utilizing timestamps to capture temporal dynamics and updating existing KGs based on these dynamics while maintaining KG consistency is a problem to be further addressed in SICKLE system.

6.2.2 Complete a Probabilistic Knowledge Graph

We have done a series experiments in Chapter 5, which combines the probabilistic scores from different KGC models and change the candidates rank position based on the final fusion score in a single round. However, in the iterative pipeline, computing new confidence scores relies on the confidence of the existing triples. In a probabilistic Knowledge Graph, each assertion or fact is associated with a probability value, indicating the degree of certainty or confidence in the truthfulness of that statement. For example, instead of simply stating that "John is a doctor," the probabilistic KG might state that "There is an 80% probability that John is a doctor." Probabilistic reasoning techniques, such as Bayesian networks [198], are applied to perform inference in a Probabilistic KG, allowing for modeling uncertain dependencies and making probabilistic inferences. It would be interesting to include probabilistic reasoning to rank the new triples in an iterative hybrid KGC pipeline.

How to infer and update probabilities in a hybrid KGC pipeline while balance computational effort, especially in large-scale KG, is a challenge to be addressed in our future work.

6.2.3 Integrate Triple Producer based on Large Language Models

Recent years, there has been a trend towards extracting triples from pre-trained language models. Instead of considering the graph structure, these works directly leverages pre-trained language Models for KG Completion. Typical works in this vein include [199, 200]. This approach differs from our literal embedding-based method in SICKLE system, where the literal serves as an additional feature to KGE.

In this context, the self-supervised Large Language Models (LLM) have been investigated for their ability to directly retrieve relational knowledge from their parameters [201], for example, through question answering, prompting using cloze-style questions [202, 203] or statement scoring [204]. Although a range of terms are used in that context, such as fact or knowledge retrieval, as well as knowledge inference, we refer to the task of accessing relational knowledge from LLM parameters as a LLM-based triple producer.

LLMs like GPT-4 have the in-context learning capability, allowing them to adapt to new tasks through careful prompt design without fine-tuning the model parameters. It would be useful to integrate and extend our SICKLE system with a LLM-based triple producer, for example, unsupervised learning triples from LLMs.

6.2.4 Updating Knowledge Graphs with Judgements

In our work, the updating of an existing KG is based on the assumption that the exiting KG is static and more trustworthy than new triples learned in KGC models. It tends to preserve the existing triples and discard the new triples that cause inconsistency. While this updating strategy is simple, it cannot handle scenarios where the dynamics nature of knowledge needs to be considered. This problem can be addressed by the ABC [205] system, which has the ability to repair Horn rules based on an ABox with the minimal changes with respect to entrenchment scores [206]. In addition, ABC outputs all possible repair solutions, considering their logical consequences. It can be used as a repairing and updating mechanisms to correct any faulty triples and rules in an expanded KG.

It would be useful to include more prudent repairing and updating mechanisms in SICKLE. In our future work, the triples that fail the consistency checking will play an important role in revising and repairing the KG with minimal changes.

6.2.5 Include Domain Specific Language Models for Log KG completion

When we apply the SICKLE system to the log KG, we train a character level language model with the FastText via the N-Gram technique to capture extra information about sub-words. The purpose is to encode entities with literal features from logs. However, the effectiveness of this encoder is not sufficient. One possible reason is that this language model focuses on encoding similar sub-words rather than considering textual context or semantic correlations.

A recent work TeleBERT [207], proposed a tele-domain pre-trained language model designed to learn the general semantic knowledge in the telecommunication field based on a more contextual Transformer structure [208]. It treats triples in knowledge graphs as textual sequences, taking entity and relation descriptions of a triple as a spliced input. Its improved version KTeleBERT [207], incorporates implicit information from log data and explicit knowledge contained in a log KG, unifying multi-source and multi-modal data. KTeleBERT also includes an adaptive numeric encoder for encoding fine-grained numerical data like indicators or attribute values. In our feature work on log KG completion, it is worth considering replacing the encoder in the literal-embedding method in SICKLE system with a more advanced KTeleBERT model encoder. This would allow us to introduce rich log literal features into the KGC pipeline.

6.3 Source Code

We have deployed the following source code:

- A system to integrate a variety of KGC methods, including embedding-based methods, literal-embedding-based methods, rule-based methods and materialisation, together with an Approximate Consistency Checking service to producing schema-consistent triples. It also equips with multiple options of negative sampling strategies. The source code is available from `https://github.com/sig4kg/SIKGC.git`. This source code has been used in an ILCC practice class to demonstrate a schema-aware KGC solution.
- A system to extract knowledge from multiple data resources, including system logs, technical documents and expert knowledge, where both rule inference and text extraction techniques are exploited and combined. The source code is available from `https://github.com/TREAT-UOE/TREAT_EXTRACTION`. This source code has been deliveried to Huawei Ltd. as a part of project deliverables.
- A system to combine multiple KGC methods with a variety of fusion methods, to analysis and understand the individual and combined KGC method's strength and weakness in link prediction task. The source code is available from `https://github.com/sig4kg/KGFusion.git`.

Appendix A

Experimental Results in Tables

The data in Table A.2, A.1, and A.3 correspond to figures in section 3.8.

Table A.2: The schema-aware silver completeness performance for NELL-995.

Triple Producers	Iteration	ε	$f_{Correctness}$	$f_{Coverage}$	$f_{Consistency}$	f_h
M	1	575,380	1.0	1.76	1.0	1.17
R	1	979,393	0.81	3.70	0.82	1.10
$\text{Base}_{\text{TransE}}$	1	670,600	0.62	2.22	0.63	0.82
$\text{Base}_{\text{SimplE}}$	1	668,487	0.61	2.21	0.62	0.81
$\text{Base}_{\text{ComplEx}}$	1	616,222	0.54	1.96	0.55	0.71
E_{TransE}	1	737,314	0.67	2.54	0.68	0.89
E_{SimplE}	1	698,764	0.61	2.36	0.63	0.82
$\text{E}_{\text{ComplEx}}$	1	653,434	0.55	2.14	0.57	0.74
L_{TransE}	1	787,723	0.74	2.78	0.74	0.98
L_{SimplE}	1	700,222	0.59	2.37	0.63	0.81
$\text{L}_{\text{ComplEx}}$	1	625,460	0.52	2.01	0.56	0.71
$\text{L}_{\text{TransE,type}}$	1	851,765	0.76	3.09	0.77	1.02
$\text{L}_{\text{SimplE,type}}$	1	874,045	0.79	3.20	0.79	1.05
$\text{L}_{\text{ComplEx,type}}$	1	836,679	0.79	3.02	0.79	1.05
$\text{E}_{\text{TransE,neg}}$	1	744,489	0.67	2.57	0.68	0.90
$\text{E}_{\text{SimplE,neg}}$	1	709,309	0.63	2.41	0.65	0.86
$\text{E}_{\text{ComplEx,neg}}$	1	629,874	0.56	2.02	0.61	0.76
$\text{L}_{\text{TransE,type,neg}}$	1	904,838	0.78	3.35	0.79	1.05
$\text{L}_{\text{SimplE,type,neg}}$	1	897,539	0.77	3.31	0.79	1.05
$\text{L}_{\text{ComplEx,type,neg}}$	1	847,007	0.79	3.07	0.80	1.06
R	2	2,008,316	0.82	8.65	0.82	1.18
$\text{Base}_{\text{TransE}}$	2	1,271,418	0.43	5.11	0.25	0.46
$\text{Base}_{\text{SimplE}}$	2	1,038,690	0.40	3.99	0.21	0.40
$\text{Base}_{\text{ComplEx}}$	2	1,027,210	0.37	3.94	0.21	0.39
E_{TransE}	2	1,269,118	0.69	5.10	0.70	0.97

Continuation of Table A.2						
Triple Producers	Iteration	ε	$f_{Correctness}$	$f_{Coverage}$	$f_{Consistency}$	f_h
E_{SimplE}	2	1,157,494	0.62	4.56	0.64	0.89
$E_{ComplEx}$	2	1,136,217	0.59	4.46	0.61	0.84
L_{TransE}	2	1,545,433	0.63	6.43	0.67	0.93
L_{SimplE}	2	1,404,820	0.55	5.75	0.67	0.93
$L_{ComplEx}$	2	1,174,789	0.76	4.65	0.59	0.80
$L_{TransE,type}$	2	1,635,268	0.67	6.86	0.79	1.10
$L_{SimplE,type}$	2	1,652,168	0.78	6.94	0.73	1.00
$L_{ComplEx,type}$	2	1,397,690	0.82	5.72	0.78	1.10
R	3	3,080,193	0.82	13.81	0.83	1.20
$Base_{TransE}$	3	2,162,641	0.33	9.39	0.16	0.32
$Base_{SimplE}$	3	1,490,708	0.31	6.16	0.12	0.25
$Base_{ComplEx}$	3	1,554,531	0.28	6.47	0.12	0.25
E_{TransE}	3	1,932,420	0.69	8.29	0.70	1.00
E_{SimplE}	3	1,533,817	0.61	6.37	0.62	0.88
$E_{ComplEx}$	3	1,549,775	0.59	6.45	0.60	0.86
L_{TransE}	3	2,731,963	0.65	12.13	0.71	0.99
L_{SimplE}	3	2,209,418	0.66	9.62	0.70	0.97
$L_{ComplEx}$	3	1,955,428	0.60	8.40	0.61	0.88
$L_{TransE,type}$	3	2,936,882	0.76	12.23	0.80	1.14
$L_{SimplE,type}$	3	2,356,820	0.71	10.33	0.72	1.02
$L_{ComplEx,type}$	3	2,824,321	0.78	12.58	0.78	1.12
R-M-L	1	1,990,646	0.87	8.57	0.88	1.24
M-R-L	1	1,998,032	0.82	8.61	0.83	1.18
R-L-M	1	1,972,090	0.81	8.48	0.82	1.17
R-M-L	2	4,001,632	0.84	18.24	0.88	1.26
M-R-L	2	4,288,686	0.82	19.62	0.85	1.23
R-L-M	2	4,074,015	0.80	18.59	0.82	1.19
R, L, M (parallel)	1	1,801,252	0.79	7.70	0.80	1.14
R, L, M (parallel)	2	4,475,505	0.82	20.52	0.82	1.21
End of Table						

Table A.3: The schema-aware silver completeness performance for TREAT dataset.

Triple Producers	Iteration	ε	$f_{Correctness}$	$f_{Coverage}$	$f_{Consistency}$	f_h
M	1	47,266	1.0	0.48	1.0	0.65
R	1	159,296	0.70	2.43	0.99	1.44

Continuation of Table A.3						
Triple Producers	Iteration	ε	$f_{Correctness}$	$f_{Coverage}$	$f_{Consistency}$	f_h
$\text{Base}_{\text{TransE}}$	1	122,227	0.84	2.07	0.96	1.10
$\text{Base}_{\text{SimplE}}$	1	119,185	0.76	1.99	0.96	1.05
$\text{Base}_{\text{ComplEx}}$	1	118,475	0.76	1.97	0.96	1.05
E_{TransE}	1	127,086	0.87	2.19	0.95	1.25
E_{SimplE}	1	118,803	0.74	1.99	0.95	1.08
$\text{E}_{\text{ComplEx}}$	1	118,745	0.74	1.98	0.94	1.08
L_{TransE}	1	130,421	0.78	2.27	0.99	1.16
L_{SimplE}	1	140,988	0.79	2.54	0.99	1.19
$\text{L}_{\text{ComplEx}}$	1	142,725	0.78	2.58	0.99	1.20
$\text{E}_{\text{TransE,neg}}$	1	135,759	0.93	2.41	0.99	1.35
$\text{E}_{\text{SimplE,neg}}$	1	133,995	0.88	2.36	0.99	1.28
$\text{E}_{\text{ComplEx,neg}}$	1	131,353	0.86	2.30	0.99	1.26
$\text{L}_{\text{TransE,neg}}$	1	131,484	0.79	2.30	0.99	1.18
$\text{L}_{\text{SimplE,neg}}$	1	174,513	0.80	3.38	0.99	1.30
$\text{L}_{\text{ComplEx,neg}}$	1	146,573	0.81	2.68	0.98	1.26
R	2	273,587	0.98	5.87	0.99	1.36
$\text{Base}_{\text{TransE}}$	2	193008	0.57	3.85	0.58	0.80
$\text{Base}_{\text{SimplE}}$	2	141872	0.43	2.56	0.45	0.61
$\text{Base}_{\text{ComplEx}}$	2	152498	0.48	2.83	0.50	0.67
E_{TransE}	2	197,944	0.92	3.97	0.96	1.26
E_{SimplE}	2	160,135	0.78	3.02	0.96	1.13
$\text{E}_{\text{ComplEx}}$	2	166,169	0.79	3.17	0.95	1.14
L_{TransE}	2	185,286	0.72	3.65	0.99	1.12
L_{SimplE}	2	193,810	0.70	3.87	0.99	1.11
$\text{L}_{\text{ComplEx}}$	2	196,330	0.69	3.93	0.99	1.11
$\text{E}_{\text{TransE,neg}}$	2	187,450	0.96	3.71	0.99	1.29
$\text{E}_{\text{SimplE,neg}}$	2	178,393	0.93	3.48	0.99	1.26
$\text{E}_{\text{ComplEx,neg}}$	2	184,899	0.91	3.64	0.99	1.26
$\text{L}_{\text{TransE,neg}}$	2	190,387	0.72	3.78	0.99	1.13
$\text{L}_{\text{SimplE,neg}}$	2	204,295	0.88	4.13	0.99	1.25
$\text{L}_{\text{ComplEx,neg}}$	2	202,907	0.83	4.10	0.99	1.22
R	3	389,204	0.98	8.78	0.99	1.40
$\text{Base}_{\text{TransE}}$	3	261,730	0.45	5.57	0.46	0.65
$\text{Base}_{\text{SimplE}}$	3	171,763	0.32	3.31	0.34	0.47
$\text{Base}_{\text{ComplEx}}$	3	186,576	0.36	3.68	0.41	0.55
E_{TransE}	3	239,195	0.94	5.01	0.97	1.30

Continuation of Table A.3

Triple Producers	Iteration	ε	$f_{Correctness}$	$f_{Coverage}$	$f_{Consistency}$	f_h
E_{SimplE}	3	173,047	0.76	3.34	0.96	1.13
$E_{ComplEx}$	3	181,280	0.78	3.55	0.96	1.15
L_{TransE}	3	227,472	0.67	4.72	0.99	1.11
L_{SimplE}	3	227,143	0.64	4.71	0.99	1.08
$L_{ComplEx}$	3	235,414	0.64	4.91	0.99	1.08
$E_{TransE,neg}$	3	242,761	0.97	5.10	0.99	1.34
$E_{SimplE,neg}$	3	192,639	0.94	3.84	0.99	1.29
$E_{ComplEx,neg}$	3	197,555	0.90	3.96	0.99	1.26
$L_{TransE,neg}$	3	229,777	0.68	4.77	0.99	1.11
$L_{SimplE,neg}$	3	216,870	0.91	4.44	0.99	1.28
$L_{ComplEx,neg}$	3	242,783	0.71	5.10	0.99	1.15
R-M-L	1	263,479	0.87	5.62	0.99	1.51
M-R-L	1	266,692	0.87	5.70	0.99	1.51
R-L-M	1	267,361	0.87	5.71	0.99	1.50
R-M-L	2	428,528	0.85	9.76	0.99	1.32
M-R-L	2	426,555	0.85	9.72	0.99	1.31
R-L-M	2	428,726	0.85	9.77	0.99	1.31
M, L, R (parallel)	1	219,923	0.85	4.52	0.98	1.43
M, L, R (parallel)	2	366,714	0.91	8.21	0.99	1.35

End of Table

Table A.1: DBped-P dataset, the schema-aware silver completeness performance of R-method, M-method, E-methods, L-methods and their combinations.

Triple Producers	Iteration	ε	$f_{Correctness}$	$f_{Coverage}$	$f_{Consistency}$	f_h
M	1	1,447,026	1.0	0.10	1.0	0.25
R	1	4,581,505	0.97	2.48	0.99	1.23
$\text{Base}_{\text{TransE}}$	1	2,848,923	0.64	1.16	0.80	0.82
$\text{Base}_{\text{SimplE}}$	1	3,548,906	0.56	1.70	0.79	0.83
$\text{Base}_{\text{ComplEx}}$	1	2,931,531	0.67	1.23	0.87	0.87
E_{TransE}	1	3,275,792	0.81	1.49	0.91	1.00
E_{SimplE}	1	2,946,067	0.65	1.24	0.91	0.87
$\text{E}_{\text{ComplEx}}$	1	3,416,227	0.83	1.59	0.89	0.92
L_{TransE}	1	3,287,436	0.82	1.49	0.98	1.03
L_{SimplE}	1	3,386,236	0.83	1.57	0.98	1.05
$\text{L}_{\text{ComplEx}}$	1	3,321,803	0.81	1.52	0.95	1.03
$\text{E}_{\text{TransE,neg}}$	1	3,413,456	0.81	1.59	0.95	1.03
$\text{E}_{\text{SimplE,neg}}$	1	3,325,336	0.77	1.52	0.94	0.99
$\text{E}_{\text{ComplEx,neg}}$	1	3,443,047	0.85	1.62	0.96	1.05
$\text{L}_{\text{TransE,neg}}$	1	3,434,348	0.84	1.61	0.99	1.07
$\text{L}_{\text{SimplE,neg}}$	1	3,612,969	0.85	1.74	0.99	1.08
$\text{L}_{\text{ComplEx,neg}}$	1	3,423,937	0.81	1.60	0.99	1.06
R	2	9,463,578	0.95	6.19	0.99	1.35
$\text{Base}_{\text{TransE}}$	2	4,559,090	0.44	2.46	0.47	0.63
$\text{Base}_{\text{SimplE}}$	2	5,318,803	0.37	3.04	0.37	0.52
$\text{Base}_{\text{ComplEx}}$	2	5,031,490	0.49	2.82	0.50	0.68
E_{TransE}	2	5,961,640	0.86	3.53	0.94	1.20
E_{SimplE}	2	6,001,799	0.85	3.56	0.86	1.04
$\text{E}_{\text{ComplEx}}$	2	5,640,944	0.88	3.28	0.95	1.21
L_{TransE}	2	6,541,701	0.87	3.98	0.99	1.24
L_{SimplE}	2	6,738,119	0.87	4.12	0.99	1.25
$\text{L}_{\text{ComplEx}}$	2	6,161,072	0.87	3.68	0.97	1.22
R-M-L	1	7,734,123	0.96	4.88	0.99	1.33
M-R-L	1	7,403,056	0.91	4.63	0.99	1.29
R-L-M	1	7,609,075	0.94	4.78	0.99	1.31
M, L, R (parallel)	1	6,237,373	0.90	3.74	0.97	1.26

Bibliography

1. Paulheim, H. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web* **8** (ed Cimiano, P.) 489–508. ISSN: 22104968. https://www.medra.org/servlet/aliasResolver?alias=iospress%7B%5C&%7Ddoi=10.3233/SW-160218 (Dec. 2016).
2. Carlson, A. *et al.* Toward an architecture for never-ending language learning. *Proceedings of the National Conference on Artificial Intelligence* **3,** 1306–1313 (2010).
3. Wiharja, K., Pan, J. Z. & Kollingbaum, M. J. Schema aware iterative Knowledge Graph completion. *Journal of Web Semantics,* 100616. ISSN: 15708268 (2020).
4. Pan, J. Z., Vetere, G., Gomez-Perez, J. M. & Wu, H. *Exploiting Linked Data and Knowledge Graphs in Large Organisations* (eds Pan, J. Z., Vetere, G., Gomez-Perez, J. M. & Wu, H.) ISBN: 978-3-319-45652-2. http://link.springer.com/10.1007/978-3-319-45654-6 (Springer International Publishing, Cham, 2017).
5. *Exploiting Linked Data and Knowledge Graphs for Large Organisations* (eds Pan, J. Z., Vetere, G., Gomez-Perez, J. & Wu, H.) (Springer, 2017).
6. Pan, J. *et al. Reasoning Web: Logical Foundation of Knowledge Graph Construction and Querying Answering* (Springer, 2017).
7. Guha, R., McCool, R. & Miller, E. *Semantic search* in *WWW '03: Proceedings of the 12th international conference on World Wide Web* (2003), 700–709.
8. Pan, J. Z., Taylor, S. & Thomas, E. *Reducing Ambiguity in Tagging Systems with Folksonomy Search Expansion* in *the Proc. of the 6th European Semantic Web Conference (ESWC2009)* (2009).
9. Nguyen, D. Q., Vu, T., Nguyen, T. D., Nguyen, D. Q. & Phung, D. Q. *A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalization.* in *NAACL-HLT (1)* (NAACL-HLT, 2019), 2180–2189.
10. Gu, Y. *et al. Relevance Search over Schema-Rich Knowledge Graphs* in *Proc. of the 12th ACM International WSDM Conference (WSDM2019)* (2019), 114–122.
11. Wang, H., Zhang, F., Xie, X. & Guo, M. *DKN: Deep knowledge-aware network for news recommendation* in *Proceedings of the 2018 world wide web conference* (2018), 1835–1844.
12. Wang, X. *et al. Explainable reasoning over knowledge graphs for recommendation* in *Proceedings of the AAAI conference on artificial intelligence* **33** (2019), 5329–5336.
13. Tu, K. *et al. Conditional graph attention networks for distilling and refining knowledge graphs in recommendation* in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (2021), 1834–1843.

14. Wu, S., Sun, F., Zhang, W., Xie, X. & Cui, B. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys* **55,** 1–37 (2022).
15. Wang, X., He, X., Cao, Y., Liu, M. & Chua, T.-S. *KGAT: Knowledge Graph Attention Network for Recommendation* in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2019)* (2019), 950–958.
16. Xian, Y., Fu, Z., Muthukrishnan, S., de Melo, G. & Zhang, Y. *Reinforcement Knowledge Graph Reasoning for Explainable Recommendation* in *Proceedings of SIGIR* (2019), 285–294.
17. Yang, Y., Huang, C., Xia, L. & Li, C. *Knowledge Graph Contrastive Learning for Recommendation* in *Proceedings of SIGIR* (2022), 1434–1443.
18. Wu, H. *et al. Knowledge Driven Phenotyping* in *Proc. of Medical Informatics Europe (MIE 2020)* (2020), 1327–1328.
19. Tripodi, I. J. *et al.* Applying knowledge-driven mechanistic inference to toxicogenomics. *Toxicology in Vitro* (2020).
20. Zhang, R. *et al.* Drug Repurposing for COVID-19 via Knowledge Graph Completion. *Journal of Biomedical Informatics* **115** (2021).
21. Zeng, X., Tu1, X., Liu, Y., Fu, X. & Su, Y. Toward better drug discovery with knowledge graph. *Current Opinion in Structural Biology* **72,** 114–126 (2022).
22. Deng, S. *et al. Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network* in *Proc. of the World Wide Web Conference (WWW 2019)* (2019), 678–685.
23. Cheng, D., Yang, F., Wang, X., Zhang, Y. & Zhang, L. *Knowledge Graph-based Event Embedding Framework for Financial Quantitative Investments.* in *SIGIR* (2020), 2221–2230.
24. Zhu, X. *et al.* Intelligent financial fraud detection practices in post-pandemic era. *The Innovation* **2** (4 2021).
25. Xu, H. & Giunchiglia, F. SKO Types: an entity-based scientific knowledge objects metadata schema. *Journal of Knowledge Management* **19,** 60–70 (2015).
26. Auer, S. *et al. Towards a Knowledge Graph for Science* in *Proc. of the 8th International Conference on Web Intelligence, Mining and Semantics (WIMS 2018)* (2018), 1327–1328.
27. Edelstein, E., Pan, J. Z., Soares, R. & Wyner, A. Knowledge-Driven Intelligent Survey Systems Towards Open Science. *New Generation Computing,* 397–421 (2020).
28. Pan, J. Z., Edelstein, E., Bansky, P. & Wyner, A. A Knowledge Graph Based Approach to Social Science Surveys. *Data Intell.* **3,** 477–506 (2021).
29. Kelley, A. & Garijo, D. A framework for creating knowledge graphs of scientific software metadata. *Quant. Sci. Stud.* **2,** 1423–1446 (4 2021).

30. Liang, S., Zhu, A., Zhang, J. & Shao, J. Hyper-node relational graph attention network for multi-modal knowledge graph completion. *ACM Transactions on Multimedia Computing, Communications and Applications* **19,** 1–21 (2023).
31. Xu, C. *et al. Adversarial Incomplete Multi-view Clustering.* in *IJCAI* **7** (2019), 3933–3939.
32. Rospocher, M. *et al.* Building event-centric knowledge graphs from news. *J. Web Semant.* **37-38** (2016).
33. Pan, J. Z. *et al. Content based Fake News Detection Using Knowledge Graphs* in *Proc. of the International Semantic Web Conference (ISWC2018)* (2018), 669–683.
34. Abu-Salih, B., Al-Tawil, M., Aljarah, I., Faris, H. & Wongthongtham, P. Relational Learning Analysis of Social Politics using Knowledge Graph Embedding. *Data Mining and Knowledge Discovery,* 1497–1536 (2021).
35. Liu, J. *et al.* DTN: Deep triple network for topic specific fake news detection. *J. Web Semant.* **70** (2021).
36. Phil, T. *et al. Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering* W3C Working Draft Working Group Note 2006/02/11 (2006).
37. Holger, K. *et al. A Semantic Web Primer for Object-Oriented Software Developers* W3C Working Group Note 9 March 2006 (W3C, 2006).
38. *Ontology-Driven Software Development* (eds Pan, J. Z., Staab, S., Aßmann, U., Ebert, J. & Zhao, Y.) ISBN: 978-3-642-42895-1 (Springer, 2013).
39. *Semantic Web Enabled Software Engineering* (eds Pan, J. Z. & Zhao, Y.) ISBN: 978-1-61499-369-8 (IOS Press, 2014).
40. Xie, C., Yu, B., Zeng, Z., Yang, Y. & Liu, Q. Multilayer Internet-of-Things Middleware Based on Knowledge Graph. *IEEE Internet Things J.* **8,** 2635–2648 (4 2021).
41. Althar, R. R. & Samanta, D. The realist approach for evaluation of computational intelligence in software engineering. *Innov. Syst. Softw. Eng.* **17,** 17–27 (1 2021).
42. Bader, S. R., Grangel-González, I., Nanjappa, P., Vidal, M.-E. & Maleshkova, M. *A Knowledge Graph for Industry 4.0* in *Proceedings of the 17th Extended Semantic Web Conference (ESWC 2020)* (2020), 465–480.
43. Buchgeher, G., Gabauer, D., Gil, J. M. & Ehrlinger, L. Knowledge Graphs in Manufacturing and Production: A Systematic Literature Review. *IEEE Access* **9** (2021).
44. Cai, H., Zheng, V. W. & Chang, K. C. C. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Transactions on Knowledge and Data Engineering* **30,** 1616–1637. ISSN: 15582191. arXiv: 1709.07604 (2018).
45. Ji, S., Pan, S., Cambria, E., Marttinen, P. & Philip, S. Y. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE transactions on neural networks and learning systems* **33,** 494–514 (2021).

46. Chen, X., Jia, S. & Xiang, Y. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications* **141.** ISSN: 09574174 (2020).
47. Hur, A., Janjua, N. & Ahmed, M. A Survey on State-of-the-art Techniques for Knowledge Graphs Construction and Challenges ahead. arXiv: 2110.08012. http://arxiv.org/abs/2110.08012 (2021).
48. Rossi, A., Barbosa, D., Firmani, D., Matinata, A. & Merialdo, P. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data* **15.** ISSN: 1556472X. arXiv: 2002.00819 (2021).
49. Zhang, W. *et al.* Knowledge Graph Reasoning with Logics and Embeddings: Survey and Perspective. arXiv: 2202.07412. http://arxiv.org/abs/2202.07412 (2022).
50. Xu, C. *et al.* Uncertainty-Aware Multiview Deep Learning for Internet of Things Applications. *IEEE Transactions on Industrial Informatics* **19,** 1456–1466 (2023).
51. A survey on deep learning based knowledge tracing. *Knowledge-Based Systems* **258,** 110036. ISSN: 0950-7051. https://www.sciencedirect.com/science/article/pii/S0950705122011297 (2022).
52. Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J. & Yakhnenko, O. Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems,* 1–9 (2013).
53. Sun, Z., Deng, Z. H., Nie, J. Y. & Tang, J. Rotate: Knowledge graph embedding by relational rotation in complex space. *7th International Conference on Learning Representations, ICLR 2019,* 1–18. arXiv: 1902.10197 (2019).
54. Tran, H. D., Stepanova, D., Gad-Elrab, M. H., Lisi, F. A. & Weikum, G. Towards nonmonotonic relational learning from knowledge graphs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **10326 LNAI,** 94–107. ISSN: 16113349 (2017).
55. Meilicke, C., Chekol, M. W., Ruffinelli, D. & Stuckenschmidt, H. Anytime bottom-up rule learning for knowledge graph completion. *IJCAI International Joint Conference on Artificial Intelligence* **2019-Augus,** 3137–3143. ISSN: 10450823 (2019).
56. Fang, U., Li, J., Akhtar, N., Li, M. & Jia, Y. GoMIC: Multi-view image clustering via self-supervised contrastive heterogeneous graph co-learning. *World Wide Web.* ISSN: 1573-1413. https://doi.org/10.1007/s11280-022-01110-6 (Oct. 2022).
57. Yang, S. *et al.* Robust cross-network node classification via constrained graph mutual information. *Knowledge-Based Systems* **257,** 109852. ISSN: 0950-7051. https://www.sciencedirect.com/science/article/pii/S0950705122009455 (2022).
58. Fang, U., Li, J., Lu, X., Mian, A. & Gu, Z. Robust image clustering via context-aware contrastive graph learning. *Pattern Recognition* **138,** 109340. ISSN: 0031-3203. https://www.sciencedirect.com/science/article/pii/S0031320323000419 (2023).

59. Pan, W., Wei, W. & Mao, X.-L. Context-aware Entity Typing in Knowledge Graphs, 2240–2250. arXiv: 2109.07990 (2021).
60. Ge, X., Wang, Y.-C., Wang, B. & Kuo, C.-C. J. Core: A Knowledge Graph Entity Type Prediction Method Via Complex Space Regression and Embedding. *SSRN Electronic Journal.* ISSN: 0167-8655. https://doi.org/10.1016/j.patrec.2022.03.024 (2022).
61. Chen, J. *et al.* OWL2Vec*: embedding of OWL ontologies. *Machine Learning* **110,** 1813–1845. ISSN: 15730565. arXiv: 2009.14654 (2021).
62. Fionda, V. & Pirrò, G. Fact checking via evidence patterns. *IJCAI International Joint Conference on Artificial Intelligence* **2018-July,** 3755–3761. ISSN: 10450823 (2018).
63. Fionda, V. & Pirró, G. Triple2Vec: Learning Triple Embeddings from Knowledge Graphs. *Aaai.* arXiv: 1905.11691. http://arxiv.org/abs/1905.11691 (2019).
64. Lv, X., Hou, L., Li, J. & Liu, Z. Differentiating concepts and instances for knowledge graph embedding. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018,* 1971–1979. ISSN: 2331-8422. arXiv: 1811.04588 (2018).
65. D'Amato, C., Quatraro, N. F. & Fanizzi, N. Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **12731 LNCS,** 441–457. ISSN: 16113349 (2021).
66. Bhatia, S., Dwivedi, P. & Kaur, A. Tell Me Why Is It So? Explaining Knowledge Graph Relationships by Finding Descriptive Support Passages. arXiv: 1803.06555. http://arxiv.org/abs/1803.06555 (2018).
67. Pirrò, G. Fact-checking via path embedding and aggregation. *CEUR Workshop Proceedings* **2722,** 149–158. ISSN: 16130073. arXiv: 2011.08028 (2020).
68. Du, J. *et al.* Validation of Growing Knowledge Graphs by Abductive Text Evidences. *Proceedings of the AAAI Conference on Artificial Intelligence* **33,** 2784–2791. ISSN: 2159-5399 (2019).
69. Gad-Elrab, M. H., Urbani, J., Stepanova, D. & Weikum, G. Exfakt: A framework for explaining facts over knowledge graphs and text. *WSDM 2019 - Proceedings of the 12th ACM International Conference on Web Search and Data Mining,* 87–95 (2019).
70. Wiharja, K., Pan, J. Z., Kollingbaum, M. & Deng, Y. More is better: Sequential combinations of knowledge graph embedding approaches. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **11341 LNCS,** 19–35. ISSN: 16113349 (2018).
71. Wang, F. *et al.* LEKG: A System for Constructing Knowledge Graphs from Log Extraction. *ACM International Conference Proceeding Series,* 181–185 (2021).

72. Ahmadi, N., Huynh, V. P., Meduri, V., Ortona, S. & Papotti, P. Mining Expressive Rules in Knowledge Graphs. *Journal of Data and Information Quality* **12.** ISSN: 19361963 (2020).
73. Loster, M. *et al.* Few-shot knowledge validation using rules. *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021,* 3314–3324 (2021).
74. Wang, Z., Zhang, J., Feng, J. & Chen, Z. Knowledge graph embedding by translating on hyperplanes. *Proceedings of the National Conference on Artificial Intelligence* **2,** 1112–1119. ISSN: 2159-5399 (2014).
75. Jiang, F., Drummond, T. & Cohn, T. *Don't Mess with Mister-in-Between: Improved Negative Search for Knowledge Graph Completion* in *EACL 2023 - 17th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference* (2023), 1810–1824. ISBN: 9781959429449.
76. Hu, K., Liu, H. & Hao, T. *A Knowledge Selective Adversarial Network for Link Prediction in Knowledge Graph* in *Natural Language Processing and Chinese Computing* (eds Tang, J., Kan, M.-Y., Zhao, D., Li, S. & Zan, H.) (Springer International Publishing, Cham, 2019), 171–183. ISBN: 978-3-030-32233-5.
77. Niu, J., Sun, Z. & Zhang, W. *Enhancing Knowledge Graph Completion with Positive Unlabeled Learning* in (2018), 296–301.
78. Bhardwaj, P., Kelleher, J., Costabello, L. & O'Sullivan, D. Poisoning knowledge graph embeddings via relation inference patterns. *ACL-IJCNLP 2021 - 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, Proceedings of the Conference,* 1875–1888. arXiv: 2111.06345 (2021).
79. Du, J., Qi, K. & Shen, Y. Knowledge graph embedding with logical consistency. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **11221 LNAI,** 123–135. ISSN: 16113349 (2018).
80. Jain, N., Tran, T. K., Gad-Elrab, M. H. & Stepanova, D. Improving Knowledge Graph Embeddings with Ontological Reasoning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **12922 LNCS,** 410–426. ISSN: 16113349 (2021).
81. Zhang, W. *et al.* Iteratively learning embeddings and rules for knowledge graph reasoning. *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019,* 2366–2377. arXiv: 1903.08948 (2019).
82. Chen, Z. *et al.* Knowledge graph completion: A review. *IEEE Access* **8,** 192435–192456. ISSN: 21693536 (2020).
83. Shi, B. & Weninger, T. Open-world knowledge graph completion. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018,* 1957–1964. arXiv: 1711.03438 (2018).

84. Xie, R., Liu, Z., Jia, J., Luan, H. & Sun, M. Representation learning of knowledge graphs with entity descriptions. *30th AAAI Conference on Artificial Intelligence, AAAI 2016,* 2659–2665 (2016).
85. Xiao, H., Huang, M., Meng, L. & Zhu, X. SSP: Semantic space projection for knowledge graph embedding with text descriptions. *31st AAAI Conference on Artificial Intelligence, AAAI 2017,* 3104–3110. arXiv: 1604.04835 (2017).
86. Niu, L. *et al.* Open-world knowledge graph completion with multiple interaction attention. *World Wide Web* **24,** 419–439. ISSN: 15731413 (2021).
87. Daza, D., Cochez, M. & Groth, P. *Inductive Entity Representations from Text via Link Prediction* in *Proceedings of the Web Conference 2021* **1** (ACM, New York, NY, USA, Apr. 2021), 798–808. ISBN: 9781450383127. arXiv: 2010.03496. http://arxiv.org/abs/2010.03496%7B%5C%%7D0Ahttp://dx.doi.org/10.1145/3442381.3450141%20https://dl.acm.org/doi/10.1145/3442381.3450141.
88. Gesese, G. A., Biswas, R., Alam, M. & Sack, H. A Survey on Knowledge Graph Embeddings with Literals: Which model links better Literal-ly? **0.** arXiv: 1910.12507. http://arxiv.org/abs/1910.12507 (2019).
89. Teru, K. K., Denis, E. G. & Hamilton, W. L. Inductive relation prediction by subgraph reasoning. *37th International Conference on Machine Learning, ICML 2020* **PartF16814,** 9390–9399. arXiv: 1911.06962 (2020).
90. Devlin, J., Chang, M. W., Lee, K. & Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv,* 4171–4186. ISSN: 23318422 (2018).
91. Shearer, R., Motik, B. & Horrocks, I. HermiT: A Highly-Efficient Reasoner for Description Logics. *Cs.Ox.Ac.Uk,* 1–13. http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2008/ShMH08b.pdf (2008).
92. Steigmiller, A., Liebig, T. & Glimm, B. Konclude : System Description (2014).
93. Gutiérrez-Basulto, V. & Schockaert, S. From knowledge graph embedding to ontology embedding? an analysis of the compatibility between vector space representations and rules. *Principles of Knowledge Representation and Reasoning: Proceedings of the 16th International Conference, KR 2018,* 379–388. arXiv: 1805.10461 (2018).
94. Kazemi, S. M. & Poole, D. Simple embedding for link prediction in knowledge graphs. *Advances in Neural Information Processing Systems* **2018-Decem,** 4284–4295. ISSN: 10495258. arXiv: 1802.04868 (2018).
95. Trouillon, T., Welbl, J., Riedel, S., Ciaussier, E. & Bouchard, G. Complex embeddings for simple link prediction. *33rd International Conference on Machine Learning, ICML 2016* **5,** 3021–3032. arXiv: 1606.06357 (2016).
96. Svacina, J. *et al.* On Vulnerability and Security Log analysis: A Systematic Literature Review on Recent Trends. *ACM International Conference Proceeding Series,* 175–180 (2020).

97. Du, M., Li, F., Zheng, G. & Srikumar, V. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. *Proceedings of the ACM Conference on Computer and Communications Security,* 1285–1298. ISSN: 15437221 (2017).
98. Studiawan, H., Payne, C. & Sohel, F. Graph clustering and anomaly detection of access control log for forensic purposes. *Digital Investigation* **21,** 76–87. ISSN: 17422876. http://dx.doi.org/10.1016/j.diin.2017.05.001 (2017).
99. Meng, W. *et al.* Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. *IJCAI International Joint Conference on Artificial Intelligence* **2019-Augus,** 4739–4745. ISSN: 10450823 (2019).
100. Djidjev, H., Sandine, G., Storlie, C. B. & Wiel, S. V. Graph Based Statistical Analysis of Network Traffic. *Mlg '11,* 8. https://www.cs.purdue.edu/mlg2011/papers/paper%7B%5C_%7D10.pdf (2011).
101. Ekelhart, A., Kiesling, E. & Kurniawan, K. Taming the logs – Vocabularies for semantic security analysis. *Procedia Computer Science* **137,** 109–119. ISSN: 18770509. https://doi.org/10.1016/j.procs.2018.09.011 (2018).
102. Kiesling, E., Ekelhart, A., Kurniawan, K. & Ekaputra, F. *The SEPSES Knowledge Graph: An Integrated Resource for Cybersecurity* 198–214. ISBN: 9783030307950. http://dx.doi.org/10.1007/978-3-030-30796-7%7B%5C_%7D13 (Springer International Publishing, 2019).
103. Pingle, A. *et al.* Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement. *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2019,* 879–886. arXiv: 1905.02497 (2019).
104. Yuan, D. *et al.* SherLog: Error diagnosis by connecting clues from run-time logs. *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS,* 143–154 (2010).
105. Lou, J. G., Fu, Q., Wang, Y. & Li, J. Mining dependency in distributed systems through unstructured logs analysis. *Operating Systems Review (ACM)* **44,** 91–96. ISSN: 01635980 (2010).
106. Zou, D.-Q., Qin, H. & Jin, H. UiLog: Improving Log-Based Fault Diagnosis by Log Analysis. *Journal of Computer Science and Technology* **31,** 1038–1052. ISSN: 1000-9000. http://link.springer.com/10.1007/s11390-016-1678-7 (Sept. 2016).
107. Pecchia, A., Weber, I., Cinque, M. & Ma, Y. Discovering process models for the analysis of application failures under uncertainty of event logs. *Knowledge-Based Systems* **189.** ISSN: 09507051 (2020).
108. Brandón, Á. *et al.* Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software* **159.** ISSN: 01641212 (2020).

109. He, P., Zhu, J., He, S., Li, J. & Lyu, M. R. An evaluation study on log parsing and its use in log mining. *Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016,* 654–661 (2016).
110. Pan, J. Z., Ren, Y. & Zhao, Y. Tractable approximate deduction for OWL. *Artificial Intelligence* **235,** 95–155. ISSN: 00043702 (2016).
111. Pan, J., Vetere, G., Manuel Gomez-Perez, J. & Wu, H. *Exploiting Linked Data and Knowledge Graphs in Large Organisations* English. ISBN: 978-3-319-45652-2 (Springer International Publishing Switzerland, 2017).
112. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. & Rosati, R. DL-Lite: Tractable description logics for ontologies. *Proceedings of the National Conference on Artificial Intelligence* **2,** 602–607 (2005).
113. Dong, X. *et al. Knowledge vault: A web-scale approach to probabilistic knowledge fusion* in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), 601–610.
114. Lourdusamy, R. & Abraham, S. A Survey on Methods of Ontology Learning from Text, 113–123 (2020).
115. Francisco, A. P., Baeza-Yates, R. & Oliveira, A. L. Mining query log graphs towards a query folksonomy. *Concurrency and Computation: Practice and Experience* **24,** 2179–2192. ISSN: 15320634 (2012).
116. Dawood, H. A. Graph theory and cyber security. *Proceedings - 3rd International Conference on Advanced Computer Science Applications and Technologies, ACSAT 2014,* 90–96 (2014).
117. Pei, K. *et al.* HERCULE: Attack story reconstruction via community discovery on correlated log graph. *ACM International Conference Proceeding Series* **5-9-Decemb,** 583–595 (2016).
118. Djenouri, Y., Belhadi, A. & Fournier-Viger, P. Extracting useful knowledge from event logs: A frequent itemset mining approach. *Knowledge-Based Systems* **139,** 132–148. ISSN: 09507051. https://doi.org/10.1016/j.knosys.2017.10.016 (2018).
119. Ekelhart, A., Ekaputra, F. J. & Kiesling, E. The SLOGERT Framework for Automated Log Knowledge Graph Construction. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **12731 LNCS,** 631–646. ISSN: 16113349 (2021).
120. Zhu, J. *et al.* Tools and Benchmarks for Automated Log Parsing. *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019,* 121–130. arXiv: 1811.03509 (2019).
121. Chen, X., Jia, S. & Xiang, Y. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications* **141.** ISSN: 09574174 (2020).

122. Nakashole, N., Sozio, M., Suchanek, F. M. & Theobald, M. *Query-Time Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules* in *International Workshop on Very Large Data Search* (2012). https://api.semanticscholar.org/CorpusID:9004153.

123. Ortona, S., Meduri, V. V. & Papotti, P. Robust discovery of positive and negative rules in knowledge bases. *Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018,* 1180–1191 (2018).

124. Galárraga, L. A., Teflioudi, C., Hose, K. & Suchanek, F. AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases, 413–422 (2013).

125. Galárraga, L., Teflioudi, C., Hose, K. & Suchanek, F. M. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB Journal* **24,** 707–730. ISSN: 0949877X (2015).

126. Lajus, J., Galárraga, L. & Suchanek, F. Fast and Exact Rule Mining with AMIE 3. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **12123 LNCS,** 36–52. ISSN: 16113349 (2020).

127. Meilicke, C., Chekol, M. W., Ruffinelli, D. & Stuckenschmidt, H. Anytime bottom-up rule learning for knowledge graph completion. *IJCAI International Joint Conference on Artificial Intelligence* **2019-Augus,** 3137–3143. ISSN: 10450823 (2019).

128. Halaschek-Wiener, C., Parsia, B., Sirin, E. & Kalyanpur, A. Description logic reasoning for dynamic ABoxes. *CEUR Workshop Proceedings* **189,** 200–207. ISSN: 16130073 (2006).

129. Pan, J. Z. & Horrocks, I. OWL-Eu: Adding customised datatypes into OWL. *Web Semantics* **4,** 29–39. ISSN: 15708268 (2006).

130. Krötzsch, M. Description logic rules. *European Conference on Artificial Intelligence,* 1–263 (2008).

131. Shearer, R., Motik, B. & Horrocks, I. HermiT: A highly-eficient OWL reasoner. *CEUR Workshop Proceedings* **432.** ISSN: 16130073 (2009).

132. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. & Rosati, R. Data complexity of query answering in description logics. *Artificial Intelligence* **195,** 335–360. ISSN: 0004-3702. https://www.sciencedirect.com/science/article/pii/S0004370212001257 (2013).

133. Baader, F., Brandt, S. & Lutz, C. *Pushing the EL Envelope.* in (Jan. 2005), 364–369.

134. Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A. & Katz, Y. Pellet: A practical OWL-DL reasoner. *Web Semantics* **5,** 51–53. ISSN: 15708268 (2007).

135. Thomas, E., Pan, J. Z. & Ren, Y. TrOWL: Tractable OWL 2 reasoning infrastructure. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **6089 LNCS,** 431–435. ISSN: 03029743 (2010).

136. Ji, S., Pan, S., Cambria, E., Marttinen, P. & Yu, P. S. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems,* 1–25. ISSN: 21622388. arXiv: 2002.00388 (2020).

137. Balažević, I., Allen, C. & Hospedales, T. M. Tucker: Tensor factorization for knowledge graph completion. *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference,* 5185–5194. arXiv: 1901.09590 (2019).

138. Lacroix, T., Usunier, N. & Obozinski, G. Canonical tensor decomposition for knowledge base completion. *35th International Conference on Machine Learning, ICML 2018* **7,** 4475–4486. arXiv: 1806.07297 (2018).

139. Dong, T., Wang, Z., Li, J., Bauckhage, C. & Cremers, A. B. *Triple classification using regions and fine-grained entity typing* in *Proceedings of the AAAI conference on artificial intelligence* **33** (2019), 77–85.

140. Do, P. & Phan, T. H. Developing a BERT based triple classification model using knowledge graph embedding for question answering system. *Applied Intelligence* **52,** 636–651 (2022).

141. Christian Meilicke, Patrick Betz, H. S. Why a naive way to Combine Symbolic and Latent Knowledge Base Completion works surprisingly well. *3rd Conference on Automated Knowledge Base,* 1–26 (2021).

142. Guo, S., Wang, Q., Wang, L., Wang, B. & Guo, L. Jointly embedding knowledge graphs and logical rules. *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings,* 192–202 (2016).

143. Guo, S., Ding, B., Wang, Q., Wang, L. & Wang, B. in *Communications in Computer and Information Science* 219–227 (2016). ISBN: 9789811031670. http://link.springer.com/10.1007/978-981-10-3168-7%7B%5C_%7D22.

144. Ding, B., Wang, Q., Wang, B. & Guo, L. Improving knowledge graph embedding using simple constraints. *ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)* **1,** 110–121. arXiv: 1805.02408 (2018).

145. Diaz, G., Fokoue, A. & Sadoghi, M. *EmbedS: Scalable, Ontology-aware Graph Embeddings* in (2018).

146. Gao, H., Zheng, X., Li, W., Qi, G. & Wang, M. *Cosine-Based Embedding for Completing Schematic Knowledge* in *Natural Language Processing and Chinese Computing* (eds Tang, J., Kan, M.-Y., Zhao, D., Li, S. & Zan, H.) (Springer International Publishing, Cham, 2019), 249–261. ISBN: 978-3-030-32233-5.

147. Geng, Y. *et al. OntoZSL: Ontology-enhanced zero-shot learning* in *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021* (ACM, New York, NY, USA, Apr. 2021), 3325–3336. ISBN: 9781450383127. arXiv: 2102.07339. https://dl.acm.org/doi/10.1145/3442381.3450042.

148. Kulmanov, M., Liu-Wei, W., Yan, Y. & Hoehndorf, R. EL Embeddings: Geometric construction of models for the description logic EL++. *IJCAI International Joint Conference on Artificial Intelligence* **2019-Augus,** 6103–6109. ISSN: 10450823. arXiv: 1902.10499 (2019).

149. Motik, B. *et al.* OWL 2 web ontology language profiles. *W3C recommendation* **27** (2009).

150. Kulmanov, M., Smaili, F. Z., Gao, X. & Hoehndorf, R. Semantic similarity and machine learning with ontologies. *Briefings in Bioinformatics* **22,** 1–18. ISSN: 14774054 (2021).

151. Smaili, F. Z., Gao, X. & Hoehndorf, R. Onto2vec: joint vector-based representation of biological entities and their ontology-based annotations. *Bioinformatics* **34,** i52–i60 (2018).

152. Smaili, F. Z., Gao, X. & Hoehndorf, R. OPA2Vec: combining formal and informal content of biomedical ontologies to improve similarity-based prediction. *Bioinformatics* **35,** 2133–2140 (2019).

153. Holter, O. M., Myklebust, E. B., Chen, J. & Jimenez-Ruiz, E. Embedding OWL ontologies with OWL2Vec. *CEUR Workshop Proceedings* **2456,** 33–36. ISSN: 16130073 (2019).

154. Chen, J. *et al.* Contextual semantic embeddings for ontology subsumption prediction. *World Wide Web.* ISSN: 15731413. arXiv: 2202.09791 (2023).

155. Safavi, T., Zhu, J. & Koutra, D. NEGATER: Unsupervised Discovery of Negatives in Commonsense Knowledge Bases. *EMNLP 2021 - 2021 Conference on Empirical Methods in Natural Language Processing, Proceedings,* 5633–5646. arXiv: 2011.07497 (2021).

156. Arnaout, H. & Razniewski, S. Can large language models generate salient negative statements ? *KBC-LM Workshop at ISWC* (2023).

157. Weikum, G. Enriching knowledge bases with interesting negative statements. *Automated Knowledge Base Construction* (2020).

158. Arnaout, H., Razniewski, S., Weikum, G. & Pan, J. Z. *UnCommonSense: Informative Negative Knowledge about Everyday Concepts* **1,** 37–46. ISBN: 9781450392365. arXiv: 2208.09292 (Association for Computing Machinery, 2022).

159. Arnaout, H., Razniewski, S., Weikum, G. & Pan, J. Z. Negative Knowledge for Open-world Wikidata. *The Web Conference 2021 - Companion of the World Wide Web Conference, WWW 2021* **2,** 544–551 (2021).

160. Wang, Q., Mao, Z., Wang, B. & Guo, L. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* **29,** 2724–2743. ISSN: 10414347 (2017).

161. Nguyen, D. Q. A survey of embedding models of entities and relationships for knowledge graph completion, 1–14. arXiv: 1703.08098. http://arxiv.org/abs/1703.08098 (2017).

162. Meilicke, C. *et al.* Fine-grained evaluation of rule- and embedding-based systems for knowledge graph completion. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **11136 LNCS,** 3–20. ISSN: 16113349 (2018).

163. Meilicke, C., Chekol, M. W., Ruffinelli, D. & Stuckenschmidt, H. Anytime bottom-up rule learning for knowledge graph completion. *IJCAI International Joint Conference on Artificial Intelligence* **2019-Augus,** 3137–3143. ISSN: 10450823. arXiv: arXiv:2004.04412v1 (2019).

164. Selman, B., Kautz, H. & Hill, M. nowledge Compilation Horn Approximations (1991).

165. Kautz, H. & Selman, B. Knowledge compilation and theory approximation. *Journal of the ACM* **43,** 193–224 (1996).

166. Qian, J., Li, G., Atkinson, K. & Yue, Y. Understanding Negative Sampling in Knowledge Graph Embedding. *International Journal of Artificial Intelligence & Applications* **12,** 71–81. ISSN: 09762191 (2021).

167. Pan, J. Z. & Thomas, E. Approximating OWL-DL Ontologies. *AAAI,* 1434–1439 (2007).

168. Pan, J. Z. & Horrocks, I. *Reasoning in the SHOQ(D_n) Description Logic* in *Proceedings of the 2002 International Workshop on Description Logics (DL2002)* (eds Horrocks, I. & Tessaris, S.) (2002).

169. Pan, J. Z. & Horrocks, I. *Web Ontology Reasoning with Datatype Groups* in *Proc. of the International Semantic Web Conference* (2003), 47–63.

170. Mohit, B. in *Natural language processing of semitic languages* 221–245 (Springer, 2014).

171. Qudus, U., Röder, M., Saleem, M. & Ngonga Ngomo, A.-C. HybridFC: A Hybrid Fact-Checking Approach for Knowledge Graphs. *ISWC,* 1–19. https://github.com/dice-group/HybridFC. (2022).

172. Akrami, F., Saeef, M. S., Zhang, Q., Hu, W. & Li, C. Realistic Re-evaluation of Knowledge Graph Completion Methods: An Experimental Study. *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 1995–2010. ISSN: 07308078. arXiv: 2003.08001 (2020).

173. Ali, M. *et al.* PyKEEN 1.0: A python library for training and evaluating knowledge graph embeddings. *Journal of Machine Learning Research* **22,** 1–6. ISSN: 15337928. arXiv: 2007.14175 (2021).

174. Zhu, R. *et al.* A Closer Look at Probability Calibration of Knowledge Graph Embedding, 1–6 (2022).
175. Nickel, M., Murphy, K., Tresp, V. & Gabrilovich, E. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE* **104,** 11–33 (2016).
176. Platt, J. *Probabilistic Outputs for Support vector Machines and Comparisons to Regularized Likelihood Methods* in (1999).
177. Niculescu-Mizil, A. & Caruana, R. *Predicting good probabilities with supervised learning* in (Jan. 2005), 625–632.
178. Zadrozny, B. & Elkan, C. Obtaining Calibrated Probability Estimates from Decision Trees and Naive Bayesian Classifiers. *ICML* **1** (May 2001).
179. Shen, T., Zhang, F. & Cheng, J. A comprehensive overview of knowledge graph completion. *Knowledge-Based Systems* **255,** 109597. ISSN: 09507051. https://doi.org/10.1016/j.knosys.2022.109597 (2022).
180. Sun, W. *et al.* Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agent. arXiv: 2304.09542. http://arxiv.org/abs/2304.09542 (2023).
181. Wang, F., Bundy, A. & Li, X. Schema-aware Iterative Completion for Knowledge Graphs Revisited. *WWW journal* (2023).
182. Razniewski, S. & Weikum, G. *UnCommonSense in Action ! Informative Negations for Commonsense Knowledge Bases* **1,** 3–6. ISBN: 9781450394079 (Association for Computing Machinery, 2023).
183. Liu, Y. *et al.* MMKG: Multi-modal knowledge graphs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **11503 LNCS,** 459–474. ISSN: 16113349. arXiv: 1903.05485 (2019).
184. Teach, Y. C., Ruffinelli, D., Broscheit, S. & Gemulla, R. *You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings* in *ICLR* (2020).
185. Kazemi, S. M. *et al.* Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research* **21,** 1–73. ISSN: 15337928. arXiv: 1905.11485 (2020).
186. Dasgupta, S. S., Ray, S. N. & Talukdar, P. Hyte: Hyperplane-based temporally aware knowledge graph embedding. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018,* 2001–2011 (2020).
187. Lacroix, T., Obozinski, G. & Usunier, N. Tensor Decompositions for temporal knowledge base completion, 1–12. arXiv: 2004.04926. http://arxiv.org/abs/2004.04926 (2020).
188. Liu, Y., Hua, W., Xin, K. & Zhou, X. in (eds Cheng, R., Mamoulis, N., Sun, Y. & Huang, X.) 583–598 (Springer International Publishing, Cham, 2019). ISBN: 978-3-030-34222-7. http://link.springer.com/10.1007/978-3-030-34223-4%7B%5C_%7D37.

189. Lin, L. & She, K. *Tensor Decomposition-Based Temporal Knowledge Graph Embedding* in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)* (2020), 969–975.

190. Lacroix, T., Obozinski, G. & Usunier, N. *Tensor Decompositions for temporal knowledge base completion* 2020. arXiv: 2004.04926 [stat.ML].

191. Hitchcock, F. L. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics* **6,** 164–189. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sapm192761164. https://onlinelibrary.wiley.com/doi/abs/10.1002/sapm192761164 (1927).

192. Radstok, W. & Chekol, M. *Leveraging Static Models for Link Prediction in Temporal Knowledge Graphs* 2021. arXiv: 2106.15223 [cs.LG].

193. Wang, J., Zhang, W., Chen, X., Lei, J. & Lai, X. 3DRTE: 3D Rotation Embedding in Temporal Knowledge Graph. *IEEE Access* **8,** 207515–207523 (2020).

194. Xu, C., Nayyeri, M., Alkhoury, F., Shariat Yazdi, H. & Lehmann, J. *TeRo: A Time-aware Knowledge Graph Embedding via Temporal Rotation* in *Proceedings of the 28th International Conference on Computational Linguistics* (eds Scott, D., Bel, N. & Zong, C.) (International Committee on Computational Linguistics, Barcelona, Spain (Online), Dec. 2020), 1583–1593. https://aclanthology.org/2020.coling-main.139.

195. Xu, Y. *et al. RTFE: A Recursive Temporal Fact Embedding Framework for Temporal Knowledge Graph Completion* in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (eds Toutanova, K. *et al.*) (Association for Computational Linguistics, Online, June 2021), 5671–5681. https://aclanthology.org/2021.naacl-main.451.

196. Liao, S., Liang, S., Meng, Z. & Zhang, Q. *Learning Dynamic Embeddings for Temporal Knowledge Graphs* in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining* (Association for Computing Machinery, Virtual Event, Israel, 2021), 535–543. ISBN: 9781450382977. https://doi.org/10.1145/3437963.3441741.

197. Cai, B. *et al. Temporal Knowledge Graph Completion: A Survey* in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence* (International Joint Conferences on Artificial Intelligence Organization, Aug. 2023). http://dx.doi.org/10.24963/ijcai.2023/734.

198. Ben-Gal, I. Bayesian networks. *Encyclopedia of statistics in quality and reliability* (2008).

199. Petroni, F. *et al. Language Models as Knowledge Bases?* in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (2019), 2463–2473.

200. Wang, C., Liu, X. & Song, D. Language Models are Open Knowledge Graphs, 1–30. arXiv: 2010.11967. http://arxiv.org/abs/2010.11967 (2020).

201. Pan, J. Z. *et al.* Large Language Models and Knowledge Graphs: Opportunities and Challenges. **000,** 1–30. arXiv: 2308.06374. http://arxiv.org/abs/2308.06374 (2023).

202. Heinzerling, B. & Inui, K. *Language Models as Knowledge Bases: On Entity Representations, Storage Capacity, and Paraphrased Queries* 2021. arXiv: 2008.09036 [cs.CL].

203. Sachan, D. S., Zhang, Y., Qi, P. & Hamilton, W. *Do Syntax Trees Help Pre-trained Transformers Extract Information?* 2021. arXiv: 2008.09084 [cs.CL].

204. Tamborrino, A., Pellicanò, N., Pannier, B., Voitot, P. & Naudin, L. *Pre-training Is (Almost) All You Need: An Application to Commonsense Reasoning* in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (eds Jurafsky, D., Chai, J., Schluter, N. & Tetreault, J.) (Association for Computational Linguistics, Online, July 2020), 3878–3887. https://aclanthology.org/2020.acl-main.357.

205. Li, X., Bundy, A. & Smaill, A. ABC repair system for datalog-like theories. *IC3K 2018 - Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management* **2,** 335–342 (2018).

206. Li, X., Bundy, A. & Philalithis, E. *Signature Entrenchment and Conceptual Changes in Automated Theory Repair* in *The Ninth Annual Conference on Advances in Cognitive Systems* (2021).

207. Chen, Z. *et al. Tele-Knowledge Pre-training for Fault Analysis* 2023. arXiv: 2210.11298 [cs.AI].

208. Vaswani, A. *et al.* Attention is all you need. *Advances in neural information processing systems* **30** (2017).