Statements and Declarations. The authors would like to thank Huawei for supporting the research on which this paper was based under grant CIENG4721/LSC.

Schema-aware Iterative Completion for Knowledge Graphs Revisited

Fangrong Wang[1], Alan Bundy[1], Xue Li[1], Ruiqi Zhu[1], Kwabena Nuamah[1], Lei Xu[2], Stefano Mauceri[2] and Jeff Z. Pan[1,3*]

[1]School of Informatics, the University of Edinburgh, UK.
[2]Ireland Research Center, Huawei, Ireland.
[3]Edinburgh Research Centre, CSI, Huawei, UK.

*Corresponding author(s). E-mail(s): j.z.pan@ed.ac.uk;

Abstract

Recent success of knowledge graph (KG) has spurred widespread interests in methods for the problem of *Knowledge Graph Completion* (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. Hence, schema-aware KGC seems to be way to go.

In this paper, we revisited the notion of schema-aware KGC from the aspects of sampling strategies, the type of triple producers, and the ways that different KGC methods are combined. Accordingly, we present Schema-aware Iterative Completion for KnowLEdge Graphs (SICKLE), a system to integrate a variety of KGC methods, including embedding-based methods, literal-embedding-based methods, rule-based methods and a materialisation-based methods, together with an Approximate Consistency Checking (ACC) approach to producing schema correct triples.

Our experimental results show that our approach is able to improve not only the schema-awareness of KGC in producing schema-correct triples in link prediction task but also the performance in the downstream task of type prediction. Our source code is publicly available from https://github.com/sig4kg/SIKGC.

Keywords: knowledge graph completion, approximate reasoning

1 Introduction

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 [1]. Indeed, the use of knowledge graphs [1, 2] has become popular in knowledge representation and knowledge management applications, including search [3–6], recommendation [7–13], medical informatics [14–17], finance [18–20], science [21–25], multi-modal content [26, 27], media [28–31], software engineering [32–37] and industrial domains [1, 38, 39].

Knowledge Graph Completion 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 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 [40–47]. Knowledge graph embedding approaches, such as [48, 49] (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 [50, 51], 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 [52–54]. For example, train a classifier on the features of node and inward and outward edges to infer node types [55–57].

Correctness checking in KGC can be done by various methods, including probabilistic triple classification with machine learning or statistical methods [58–61], finding explanations from existing KG or external text evidence [62–65] and consistency checking with defined, external or mined constraints [66–69]. 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 identified as correct might be inconsistent with the existing KG.

Existing KGC research often uses the silver standard method [70] 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 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% triples are consistent with the original NELL-995 dataset [71] schema and satisfy the domain and range constraints. The corresponding ratios for

the DBpedia Politics subset (DBped-P) [72] are 99.6% triples consistent with the DBpedia-2016 [1] schema and 57.8% triples satisfy the property domain and range constraints. Also, previous research [73] show that translation based methods, such as TransE, cannot properly capture simple rules; even Bilinear models, such as SimplE [74] and ComplEx [75], 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. For example, when we use TransE [48] to link the NELL-995 dataset, less than 50% of the first 10 results are consistent with its schema.

Our previous work, SIC [72], challenged the silver standard method, by proposing the notion of schema-correctness. SIC evaluated the schema-correctness level for KGC task on different types of triple producers, including a knowledge graph embedding method (E-method), a rule learning method (R-method), an ontology reasoning method for materialisation (M-method) and their combinations and iterations. SIC used an approximate consistency checking (ACC) method [66] 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, so that only the schema-correct triples will be used in the next iteration of completion.

In this paper, 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 and thus raise different research questions.

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. Our baseline system 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.

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

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 [76–81]. The literal-embedding methods (L-method), such as DKRL [76], SSP [78], Conmask [77], BLP [80] and many others, have the potential of linking novel entities into existing KG, which referred as inductive LP [82]. 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. In Section 5, our experimental results confirmed this hypothesis.

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 ACC 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. Our hypothesis is that the KGE's schema features can be enhanced by injecting new schema-correct data in iterative training.

In our strategies, the ACC module plays an important role in identifying schema-correct triples and schema-inconsistent triples from a given KG or expanded KG. We use schema-correct triples for learning and prediction and only add new schema-correct triples to a target KG. Particularly, the M-method only works on consistent KGs, hence the ACC module acts as a "clean up" step to filter out inconsistent triples from the expanded KGs. We also make use of schema-inconsistent triples in negative sampling of KGE training. Our criterion of ACC is being able to detect as many as possible inconsistency justifications from a large scale KG. In this paper, we propose a new ACC method by applying the idea of *knowledge compilation* [83, 84], by transforming the TBox in complex description logic, OWL 2 [2] into a simpler description logic, DL-Lite [85]. Based on the transformed TBox in DL-Lite, we designed and calculated inconsistent justification patterns (IJPs) with more reasoning power and more uniform forms than those in SIC. With this ACC method, we scanned the ABox to identify inconsistent triples that match the IJPs. The optimised ACC module is more effective and efficient than previous work mentioned in [66] in detecting errors in a Knowledge Graph containing millions of triples.

With all these methods, we develop a unified system called *SICKLE* for schema-aware iterative hybrid KGC. The features of SICKLE include:

- SICKLE extends the training data sampling strategy for E-methods and L-methods. With our schema-aware sampling strategy, only schema-correct triples are used for positive sampling and schema-inconsistent triples are preferentially selected for negative sampling.

[2] https://www.w3.org/TR/owl2-overview/

- At the functional level, SICKLE applies a schema-aware sampling strategy and includes L-methods, which encode entities with their literal information with pre-trained language models, before applying knowledge graph embedding (KGE) based learning.
- At the algorithm level, SICKLE optimises the ACC module to identify inconsistent subsets from a given KG.
- At the implementation level, SICKLE flexibly assembles different triple producers and ACC models in combinations and runs them in an iterative manner for both Link Prediction (LP) and Type Prediction (TP). It unifies data formats and manages data for different types of triple producers and ACC modules. Thus, users no longer need to handle complicated data processing for different triple producers.

We experimentally tested SICKLE on both community benchmarks and an industrial dataset. To analyze the schema-awareness of the triple producer pipelines in SICKLE, we followed SIC's validation method: calculating schema-consistency ratio, schema-correctness ratio, coverage and their harmonica mean on a variety of triple producer combinations. We name this validation method *schema-aware silver completeness ratios*. For E-methods and L-methods, instead of evaluating KGC using the silver-standard [70], we propose a *schema-aware silver standard*, which splits a portion of test data from the original data set and extends it with new triples that materialised from the given KG, then the evaluation matrix for LP and TP are calculated based on the extended test set. Our main findings are as follows:

1. The schema-aware sampling strategy improves the schema-aware silver completeness ratios of the E-methods and the L-methods. It also has a positive effect in the downstream TP task.
2. The L-methods outperformed the E-methods in producing schema-correct triples, given high quality literal features and pre-trained language models.
3. The E-methods outperformed the L-methods in transductive LP, but if we apply the entity representations learned in LP to the downstream TP task, the L-methods outperforms the E-methods. It indicates that the L-methods encode additional schema-related features from literal while learning KGE, and these additional features are transferable to the downstream TP task.
4. The embeddings learned by iterative pipeline performs better in downstream TP tasks than embedding trained by single round. Furthermore, the combinations of different types of triple producer outperformed single triple producers on all schema-aware silver completeness ratios.
5. Even with one iteration, parallel pipeline runner approaches perform worse than serial pipeline runner approaches in terms of productivity; however, with the number of iteration increasing, the parallel approaches catch up. Furthermore, parallel approaches are significantly more efficient.

2 Background

2.1 Schema-aware KGC

We define a knowledge graph $\mathcal{G} = (\mathcal{T} \cup \mathcal{C}) \cup \mathcal{A}$, as consisting of a schema $(\mathcal{T} \cup \mathcal{C})$ and $\mathcal{A}$, which is a set of interconnected typed entities and their attributes (or ABox in description logic terminology). The schema of a KG consists of a set of ontological axioms $\mathcal{T}$(or TBox in description logic terminology) that defines the vocabulary used in a KG, as well as a set of constraints $\mathcal{C}$. 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 $\langle a, rdf : type, C \rangle$.

Given a knowledge graph $\mathcal{G} = (\mathcal{T} \cup \mathcal{C}) \cup \mathcal{A}$, SICKLE uses schema $(\mathcal{T} \cup \mathcal{C})$ to supplement a set of schema-correct triples for the data sub-graph $\mathcal{A}$. The previous work [66, 72] 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 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 $\mathcal{G} = (\mathcal{T} \cup \mathcal{C}) \cup \mathcal{A}$, a triple (h, r, t), where h and t are entities and r is a object property in $\mathcal{G}$, 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. $\mathcal{G}$ if:

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

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

Definition 2 Given a Knowledge Graph $\mathcal{G} = (\mathcal{T} \cup \mathcal{C}) \cup \mathcal{A}$, a triple (h, r, t), where h and t are entities and r is an object property in $\mathcal{G}$, 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 w.r.t. $\mathcal{G}$ if:

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

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

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

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

Our core objective is to produce "correct" triples as many as possible with respect to the schema of a Knowledge Graph. 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 5.

2.2 Combined and Iterative KGC

Two of the most common KGC methods are the KG embedding (E-method) and the logic rule learning (R-method). As discussed in [41], 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 was reviewed and analyzed in [40–42, 86–88]. 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.

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 [87, 89] and distributions of the training data [90]. [90] 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 rule 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 [89]. 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 [68, 91, 92].

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 $(\mathcal{T} \cup \mathcal{C}, \mathcal{A})$ in $\mathcal{G}$ are materialised as new axioms. In [93], the materialisation is defined as below:

Definition 6 (Materialisation) For an ontology $\mathcal{O}$, its ontology materialisation is the set $\{A \sqsubseteq B \mid A, B \in N_C, \mathcal{O} \models A \sqsubseteq B\} \cup \{a : A \mid A \in N_C, a \in N_I, \mathcal{O} \models a :$

$A\} \cup \{(a,b) : r \mid r \in N_P, a, b \in N_I, \mathcal{O} \models (a,b) : r\}$, where N_C, N_P and N_I are named classes, named properties, and individuals in $\mathcal{O}$.

Considered the differences between KGC methods, many existing work [90, 94], including our baseline system SIC [72], combined different types of KGC methods together, so that they can benefit and complement each other. IterE [90] combined embedding learning and rule learning for link prediction, in which rules are learned from embeddings and embeddings are learned from existing triples and new triples inferred by rules. Recent work [94] combined the outcome of rule based LP and embedding based LP 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. Our baseline system SIC combined a E-method, a R-method and a M-method to generate new triples, and make sure the produced triples are schema-correct, also it tends to produce as many schema-correct triples as possible in an iterative manner. Our combination method is similar to SIC, in that we make use of KG schema and combine multiple types of KGC methods to produce schema-correct triples. But we consider different strategies to improve the KGC pipelines in producing schema-correct triples. The details of our strategies are described in next section.

3 Our Approach

3.1 Problem Definition

As mentioned earlier, our core objective is to produce schema-correct triples as many as possible with respect to the schema of Knowledge Graph. Although the principle of combining multiple models into an ensemble to produce schema-correct triples has been studied in SIC, we target this objective by considering different strategies and raising new research questions:

- 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 different the execution setting 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, another is TP. 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 predict 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 advantage in downstream TP task.

3.2 Schema-aware Sampling

The embedding-based KGC approaches focus on learning low-dimensional embeddings for triple prediction. Conventional KGE methods, such as TransE [48], ComplEx [75], SimplE [74] 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 [95]. 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 [60]. However, this does not work under the open-world assumption, because absent triples could possibly be false negatives [61]. 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.

Many existing works have tackled the false negative problem [95]. For example, TransR [96] applies different probabilities in head and tail replacement to address the issue of false negatives. It gives more chance of replacing the

head in 1-to-many relations and the tail in many-to-1 relations, but it doesn't consider the schema like us. TRANSROWL [61] exploits schema in RDFS and OWL to generate a selection of negative samples in the pre-processing step for KGE training. Our method is different in that we generate negative samples during the training process, and we also take into account the top ranked schema-inconsistent predictions in an iterative pipeline, so that the new training is able to specifically target the weaknesses of the previously trained model. The existing work [97] is the closest method to our schema-aware sampling strategy, in that it leverages schema to generate inconsistent triples as negative examples and leveraging inconsistent predictions to derive negative samples in an iterative training. It's iterative training is exploited for the generation of negative samples. Our schema-aware sampling strategy differs from [97] in that our iterative KGC pipeline is to iteratively produce schema-correct triples, and top ranked schema-inconsistent predictions are the byproduct of our KGC pipeline, so it's natural to make use of them. Our baseline system SIC [72] tackled the false positive problem by applying consistency checking before and after the training procedure, but it ignored the false negative problem. In SIC's pipeline setting, only schema-correct triples were used as the training set, while the inconsistent triples were abandoned.

Our schema-aware sampling strategy, on the one hand, uses schema-correct triples as positive examples; on the other hand, we dynamically generate inconsistent negative examples, consisting of the schema-inconsistent triples predicted in the previous round and random schema-inconsistent triples.

Our schema-aware sampling strategy involves two subsets of an expanded ABox $\mathcal{A}'$: *the schema-inconsistent subset* ($\mathcal{S}_{incon}$) and *the schema-correct subset* ($\mathcal{S}_{cor}$). Based on Definition 2 - 5, these two sets are computed by equation (1) and (2), respectively.

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

$$\mathcal{S}_{incon} = \{r(h,t) \mid \exists \pi.\ r(h,t) \dot{\in} (\pi, \mathcal{G}) \wedge (\mathcal{G}' \vdash_\pi \bot)\} \tag{1}$$

It's worth noting, the equation (1) can apply to the original KG, if we regard the original ABox as $\mathcal{A}'$ and $\emptyset$ as $\mathcal{A}$, 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 $\mathcal{S}_{cor}$ are the ones not in the schema-inconsistent subset $\mathcal{S}_{incon}$ and also satisfy the constraints $\mathcal{C}$, which are domain and range constraints in our case.

$$\mathcal{S}_{cor} = \{r(h,t) \mid r(h,t) \in (\mathcal{A}' \setminus \mathcal{S}_{incon}) \wedge C_h \equiv D_r \wedge C_t \equiv R_r\} \tag{2}$$

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

Definition 7 (Schema-aware Positive Sampling) Given a knowledge graph $\mathcal{G} = (\mathcal{T} \cup \mathcal{C}) \cup \mathcal{A}'$, we use the schema-correct subset $\mathcal{S}_{cor}$ in $\mathcal{A}'$ 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 last round to do all the negative sampling, especially when the specified number of negative samples is much more than positive samples in the KGE model training. In that case, we need 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 $\mathcal{S}_{cor}$ and the schema $\mathcal{T}$, for each batch of triples $B_{pos} \subseteq \mathcal{S}_{cor}$, we randomly generate a set of schema-inconsistent negative examples:

$$B_{incon} = \{r(h', t') \mid \mathcal{T} \cup \mathcal{S}_{cor} \models r(h', t') \sqsubseteq \bot\} \tag{3}$$

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

Example 1 Given two relation assertions: α_1, α_2, which are in $\mathcal{A}'$ but not $\mathcal{A}$, and the TBox and ABox: $\mathcal{T} \cup \mathcal{A}'$, and the schema-correct subset of the ABox: $\mathcal{S}_{cor}$, where:

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

In this example, we can conclude that α_1 and α_1 belong to $\mathcal{S}_{incon}$. Thus, We can not add $\{\alpha_1, \alpha_2\}$ into the target KG, because they complete a proof of inconsistency, w.r.t. item 3 in Example 1. However, neither α_1 or α_1 belongs to

B_{incon}, w.r.t. 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 $\mathcal{S}_{incon}$ of top ranks from previous round predictions and select them as negative samples together with the generated inconsistent negative examples.

With $\mathcal{S}_{incon}$ 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 $(\mathcal{S}_{incon} \cup B_{incon})$.

The definitions in this section formalised a few key steps of algorithms in Section 4.2 and Section 3.5. Also, $\mathcal{S}_{cor}$ computed by equation (2) are incorporated in the evaluation in Section 5.

3.3 Literal-embedding based Method

In SICKLE, we include a literal-embedding model, referred to as L-method. Most of the embedding-based methods discussed in [41, 42, 88] are structure based embeddings, which do not make use of any literal information about the entities. In fact, literals in a KG encode additional information which is not captured by the entities or relations. Typical types of literals are text literals, numeric literals, image literals and external information linked to entities [81]. Recent works [76–80] have begun to explore the use of textual descriptions available in knowledge graphs to learn vector representations of entities in order to perform LP. They pointed out that literals can bring advantages to pure structure-based KGC by enriching the representation of entities and relations with semantic information. For example, literals are a natural substitute for missing topological features of novel entities or disconnected entities, hence they can help with predicting links for novel entities [77].

Following [80], the input KG of the L-methods is defined as a tuple $\mathcal{G} = (N_I, N_P, Tr, \mathcal{D})$, consisting of a set of individuals N_I, a set of named properties N_P, schema-correct triples S_{cor}, and entity descriptions $\mathcal{D}$. For each entity $e_i \in N_I$, there exists a description $d_{ei} = (w_1, ..., w_n) \in \mathcal{D}$, where all w_i are words in a vocabulary V. Given a Knowledge Graph $\mathcal{G} = (N_I, N_P, Tr, \mathcal{D})$, the L-methods are able to complete $\mathcal{G}$ 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 $\mathcal{G}$ where N_I' is an entity superset.

Modern language models, such as BERT [98], 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 on whether the literal information would benefit the ontological schema-awareness. In SICKLE, 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.

3.4 Combined and Iterative KGC

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-hot" index 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 ran after each of these producer modules to identify schema-correct subset and 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 M-method are already schema-correct.

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

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 1 and Figure 2.

In the combined and iterative pipeline, the new schema-correct subsets are incorporated into training and producing procedures. The KGEs learned from

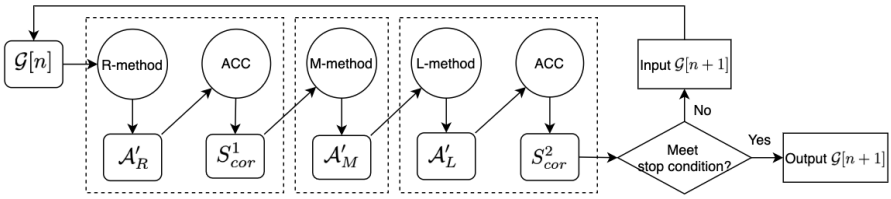

Fig. 1 An example of series iterative pipeline with R-methods, M-method and L-methods.

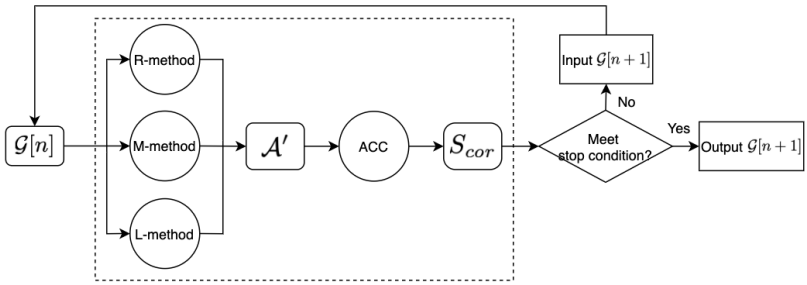

Fig. 2 An example of parallel iterative pipeline with R-methods, M-method and L-methods.

a combined and iterative pipeline are trained with more schema-correct triples from different triple producers and from iterations. Our hypothesis is that the KGE learned with this data augmentation would contain more schema-related features, hence can produce more schema-correct triples than KGE 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 Strategy for Approximate Consistency Checking

The conventional reasoning services for Description Logic ontologies, are usually expensive, due to the high reasoning complexity [93]. The consistency checking services provided by OWL reasoners, such as HermiT [99] and TrOWL [100], are relatively fast to detect if a given KG is consistent or not, however, the justification service that identify 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 [66]. This approach learns a few inconsistent justification patterns (IJPs) from TBox, and then uses the patterns to justify inconsistent subsets in ABox. However, one drawback of ACC described in [66] 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 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 paper 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 [66], is not able to identify $\{has\_parent(Anna, John\_Lewis), works\_for(Mary, John\_Lewis)\}$ as an 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 schema-inconsistent subset. Then we can justify $\{has\_parent(Anna, John\_Lewis), works\_for(Mary, John\_Lewis)\}$ 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 [83, 84] by semantically approximating a source ontology $\mathcal{O}_1$ in a more expressive DL $\mathcal{L}_1$ (source language OWL 2) with its least upper-bound $\mathcal{O}_2$ in a less expressive DL $\mathcal{L}_2$ (target language DL-Lite). In

our approach, we only apply the knowledge compilation on the TBox, namely *TBox transformation* [93].

DL-Lite [85] 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 $Domain(r)$, $Range(r)$, $InverseOf(r1, r2)$, $SubpertyOf(r1, r2)$, $DisjointWith(C1, C2)$, 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 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 4.3.

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 [93]. In a knowledge graph, the size of the ABox is usually much bigger than the size of TBox[3], so ABox reasoning is generally more complex on a larger scale than that for the TBox [93]. 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 [93], but our TBox reasoning is performed only once, and can be performed

[3]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.

Table 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	$A_1 \sqcap \exists R_x^- \sqsubseteq \bot$	$\langle e1, rdf : type, A_1\rangle, \langle e2, r_x, e1\rangle$

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.

4 Implementation

4.1 Overall Architecture

Figure 3 illustrates the overall architecture of SICKLE. We assemble pipelines with four types of triple 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 [80], 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

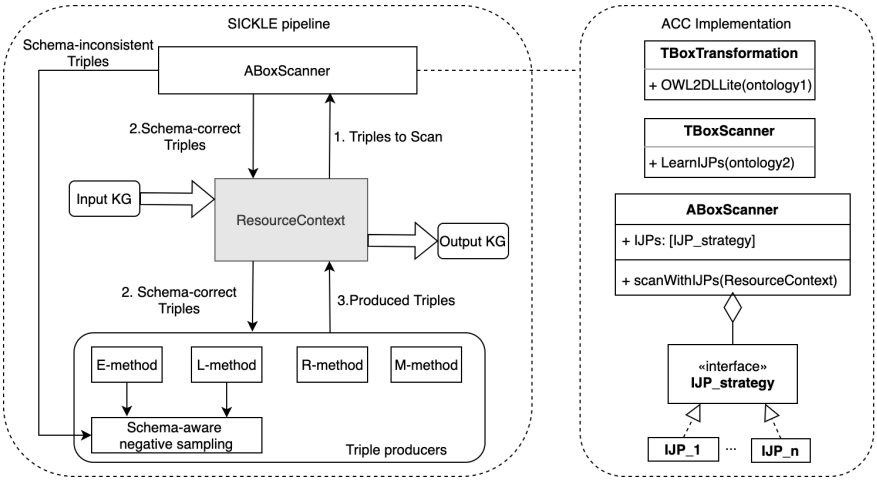

Fig. 3 Overall structure of SICKLE and an example of ACC implementation.

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 [98]; and it supports a few relational models, namely TransE [48], ComplEx [75], and SimplE [74]. 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 $\mathcal{G}' = (N_I, N_P, Tr, \mathcal{D})$, 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 [92], 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 [44], 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 [99] 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 [101]. 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 [102] to generate a schema for such a Knowledge Graph. The details of ACC implementation are described in Section 4.3.

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.

4.2 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 $\mathcal{G} = (\mathcal{T} \cup \mathcal{C}) \cup \mathcal{A}'$, SICKLE uses schema $\mathcal{T} \cup \mathcal{C}$ to supplement a set of schema-correct triples for the data sub-graph $\mathcal{A}'$. The data sub-graph $\mathcal{A}'$ 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

Algorithm 2: Schema-aware Negative Sampling

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

1 **foreach** $r(h,t) \in B_{pos}$ **do**
  `// sample n corrupted triples`
2 $B_{corrupt} \leftarrow B_{corrupt} \cup corrupt\big(r(h,t), n\big)$;
3 **end**
  `// Definition 8`
4 $B_{incon} \leftarrow identifyBatchInconsistency(\mathcal{T}, \mathcal{S}_{cor}, B_{corrupt})$;
  `// Definition 9`
5 $B_{neg} \leftarrow randomSample(B_{incon} \cup \mathcal{S}_{incon})$;

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.

4.3 ACC

According to the approach described in Section 3.5, the implementation of ACC consists of three parts:

- **TBox Transformation**: a TBox approximation module replaces the TBox $\mathcal{T}_1$ of a source ontology in a more expressive DL $\mathcal{L}_1$ (OWL 2) with its least upper-bound $\mathcal{T}_2$ in a less expressive DL $\mathcal{L}_2$ (DL-Lite).
- **TBox IJP Calculation**: a TBox scanner module infers $\mathcal{T}_2$ and calculates the ABox and TBox IJPs.
- **ABox IJP scanning**: An ABox scanner module scans ABox $\mathcal{A}$ 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 $\mathcal{O}_1$ in a more expressive DL $\mathcal{L}_1$ (source language OWL 2) with its least upper-bound $\mathcal{O}_2$ in a less expressive DL $\mathcal{L}_2$ (target language DL-Lite). In this setting, we have $\mathcal{O}_1 \models \mathcal{O}_2$ [103]. In our strategy, we only apply knowledge compilation on the TBox.

In this paper, 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 $\mathcal{T}_1$ is in OWL 2. Following [103], we use N_C, N_P to denote the set of named classes and named properties used in $\mathcal{T}_1$. For named properties in $\mathcal{T}_1$, we assign new named classes

Algorithm 3: TBox transformation from OWL2 to DL-Lite

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

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

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

$$\begin{aligned} D &= \{D_k \equiv \exists r_k, D_{k+1} \equiv \exists r_k^-\}, \\ N &= \{N_k \equiv \neg D_k, N_{k+1} \equiv \neg D_{k+1}, N_i \equiv \neg A_i\}. \end{aligned} \tag{4}$$

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 $\mathcal{T}_1$.

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: $\mathcal{T}_1 \cup D \cup N$. 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 1. Algorithm 4 illustrates how to compute pattern 1 in Table 1 by TBox reasoning.

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

Input: TBox $\mathcal{T}_2$

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 $\mathcal{T}_2$

Initialize $P_1 \leftarrow \emptyset$;
foreach $r \in N_P$ **do**
  $Set_A = \emptyset$;
  foreach $\exists r \sqsubseteq \neg A_i$ *in* $\mathcal{T}_2$ **do**
    $Set_A \leftarrow Set_A \cup \{A_i\}$;
  end
  Add (r, Set_A) into p_1 ;
end

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

Input: TBox $\mathcal{T}_2$

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 $\mathcal{T}_2$;

Initialize $P_2 \leftarrow \emptyset$;
foreach $r1 \in N_P$ **do**
  $Set_{r2} = \emptyset$;
  foreach $\exists r1 \sqsubseteq \neg \exists r2$ *in* $\mathcal{T}_2$ **do**
    $Set_{r2} \leftarrow Set_{r2} \cup \{r2\}$;
  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 [100]. TrOWL inherits the OWLAPI [4], which is more convenient for us to manipulate the ontology.

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 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: $\mathcal{S}$
Output: Schema-inconsistent subset $\mathcal{S}_{incon1}$ under p_1

Initialize $\mathcal{S}_{incon1} \leftarrow \emptyset$, $\mathcal{S}_{con1} \leftarrow \mathcal{S}$;
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**
      $\mathcal{S}_{incon1} \leftarrow \mathcal{S}_{incon1} \cup \{\langle h, r, t\rangle\}$;
    end
  end
end

Algorithm 7: ABox Scanning Strategy of p_2

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

Initialize $\mathcal{S}_{incon2} \leftarrow \emptyset$, $\mathcal{S}_{con2} \leftarrow \mathcal{S}$;
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**
      $\mathcal{S}_{incon2} \leftarrow \mathcal{S}_{incon2} \cup \{\langle h, r, t\rangle\}$;
    end
  end
end

[4] http://owlapi.sourceforge.net/

Algorithm 8: ABox Scanner for consistency checking

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

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

Algorithm 9: ABox Scanning for domain and range constraints

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

Initialize $\mathcal{S}_{con} \leftarrow \emptyset$;
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**
      $\mathcal{S}_{cor} \leftarrow \mathcal{S}_{cor} \cup \{\langle h, r, t \rangle\}$;
    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 1, which checking 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 1, which having 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 $(num_r - 1) * num_a$ times, w.r.t. 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 data analysis tool, such as Pandas [5] and its GPU extension CuDF [6].

Each IJP in Table 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, w.r.t. the equation (1) and the equation (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.

Our ACC approach is soundness-preserving syntactic approximation. Since the IJPs listed in Table 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.

5 Evaluations

5.1 Evaluation Matrix

5.1.1 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.

$$f_{Correctness} = \frac{1}{n} * \sum_{i=1}^{n} \frac{\mid \mathcal{S}_{cor}^{i} \mid}{\mid \varepsilon_i \mid} \tag{5}$$

where ε_i is the set of new triples produced by a triple producer at iteration i, $\mathcal{S}_{cor}^{i}$ 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, [72] 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:

$$f_{Coverage} = \frac{\sum_{i=1}^{n} \mid \mathcal{S}_{cor}^{i} \mid}{\mid \mathcal{Y} \mid} \tag{6}$$

where $\mathcal{Y}$ is a subset of the target Knowledge Graph $\mathcal{G}$ that consists of schema-unknown plus schema-correct triples. Larger $f_{Coverage}$ scores are an indication

[5] https://pandas.pydata.org/
[6] https://rapids.ai/start.html

that the used KGC pipeline has produced higher number of new schema-correct triples.

We define $f_{Consistent}$ as:

$$f_{Consistency} = \frac{1}{n} * \sum_{i=1}^{n} \frac{\mid \mathcal{S}_{con}^{i} \mid}{\mid \varepsilon_i \mid} \tag{7}$$

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

$$f_h = \frac{3}{\frac{1}{f_{correctness}} + \frac{1}{f_{coverage}} + \frac{1}{f_{Consistency}}} \tag{8}$$

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

5.1.2 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 [70]. 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 hit@n and MRR calculated against the test set [70].

In this paper, 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 $\mathcal{G} = (\mathcal{T} \cup \mathcal{C}) \cup \mathcal{A}$ as below:

- **Schema-aware silver standard for LP** We split the relation assertions S_{rel} in $\mathcal{A}$ 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:
  $LP_{test} = LP_{test}^{*} \cup \{r(a,b) \mid \mathcal{G} \models r(a,b), r \in N_P, a, b, \in N_I\}$
- **Schema-aware silver standard for TP** We extend the type assertions S_{type} in $\mathcal{A}$ with materialised type assertions:

$S'_{type} = S_{type} \cup \{A(a) \mid \mathcal{G} \models A(a), A \in N_C, a \in N_I\}$. 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 $\mathcal{G}$.

We evaluate LP and TP separately. We calculate hit@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 w.r.t. our datasets. Also the materialisation infers implicit entity type assertions and flattens class hierarchy.

5.2 Datasets

Our notion of schema-consistency in this paper 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 [72], we performed experiments with three Knowledge Graphs: the NELL-995 Knowledge Graph, the DBpedia politics subset, namely DBped-P [72] and a subset of a KG extracted from a network company's operation and management logs, namely TREAT [67].

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 [7], 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.

The TREAT dataset contains 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.

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%. [43] pointed out that SIC's efficacy is limited to the number of constraints furnished before running consistency

[7] http://wiki.dbpedia.org/downloads-2016-04

checking. In Table 2, we compared the $f_{correctness}$ scores of top 10 predicitons 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 Section 3.

Table 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

5.3 Experimental Setting

We assemble pipelines with four types of triple producers: the R-methods, the M-method, the E-methods and the L-methods. Under the E-method and L-method categories, we evaluated three KG embeddding models: TransE, SimplE and ComplEx. We use BERT as the text encoder in L-method for both NELL-995 and DBped-P datasets. Different from NELL-995 and DBped-P datasets, the TREAT dataset uses a domain specific FastText [8] 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 TREAT dataset contains a lot of acronyms and sub-words, and usually do not follow a natural language grammar [67]. 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.

The literal features in the L-methods are entity names and descriptions. For 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 they had their own reasons. 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. And most of the entity text in the TREAT dataset already contains type descriptions.

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

[8] https://fasttext.cc/

5.4 Empirical Results

5.4.1 Schema-aware Silver Completeness ratios

We compare the schema-aware silver completeness ratios among a variety of pipelines, w.r.t. our research objectives and strategies described in Section 3.

In Figure 4, 5 and 6, 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 run 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, obviously, 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 basic embedding models in 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 4 and Figure 5, the L-methods outperformed the E-methods in almost all schema-aware silver completeness ratios. In Figure 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.

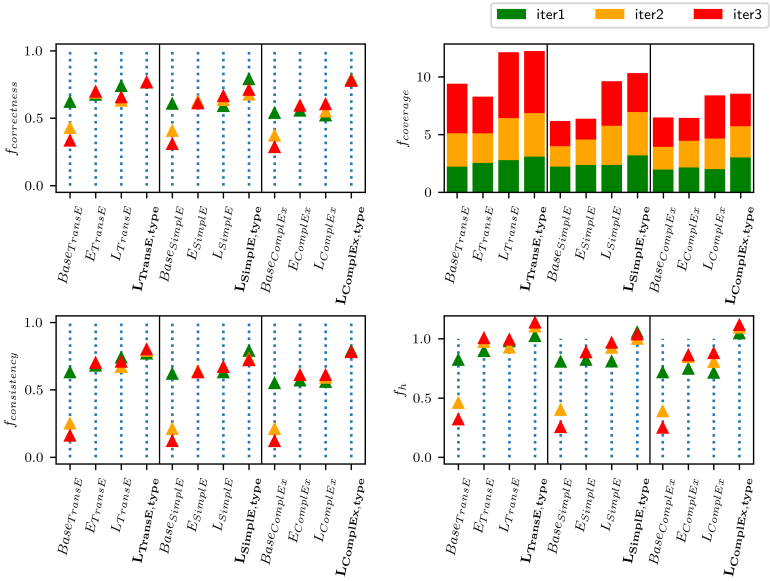

Fig. 4 NELL-995 dataset. Compare E-methods and L-methods in the schema-aware silver completeness performance

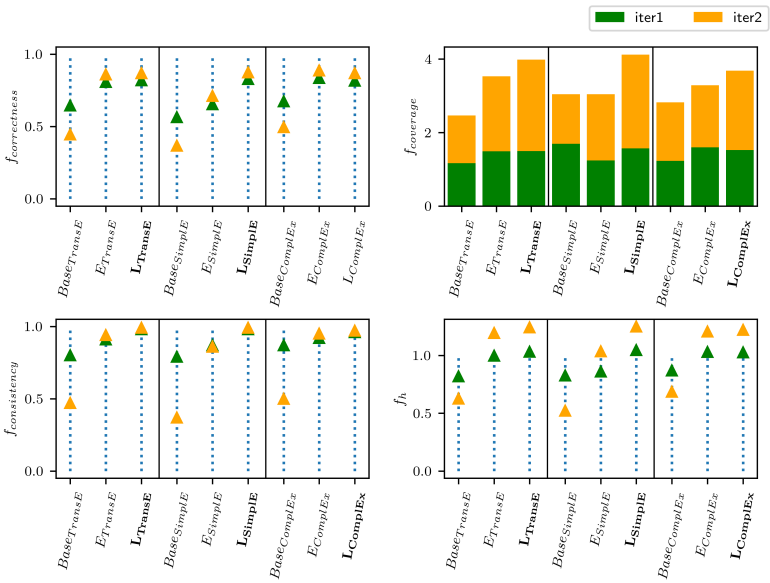

Fig. 5 DBped-P dataset. Compare E-methods and L-methods in the schema-aware silver completeness performance

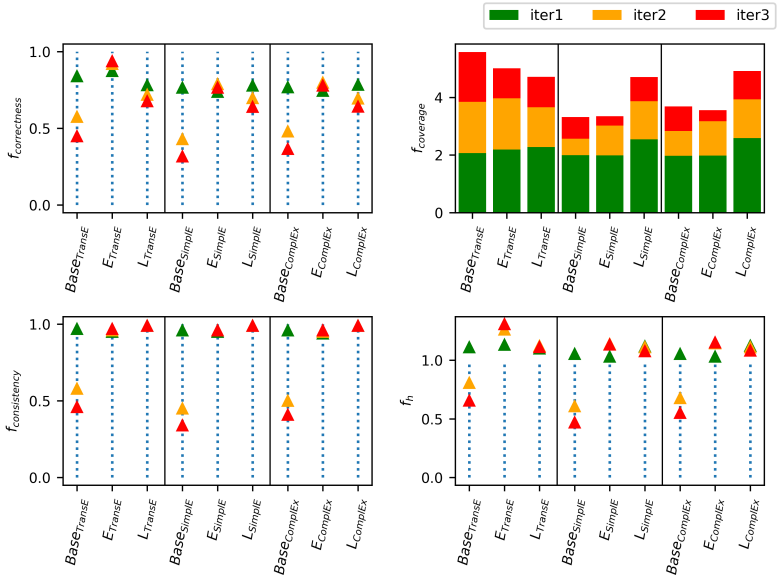

Fig. 6 TREAT dataset. Compare E-methods and L-methods in the schema-aware silver completeness performance

In Figure 7, Figure 8 and Figure 9, 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 positive impact on the performance of schema-aware silver completeness ratios, involving ontological reasoning in training procedure would significantly increase the computing resource consumed. The E-methods with schema-aware negative sampling strategy took more than 24 hours on 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 run 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.

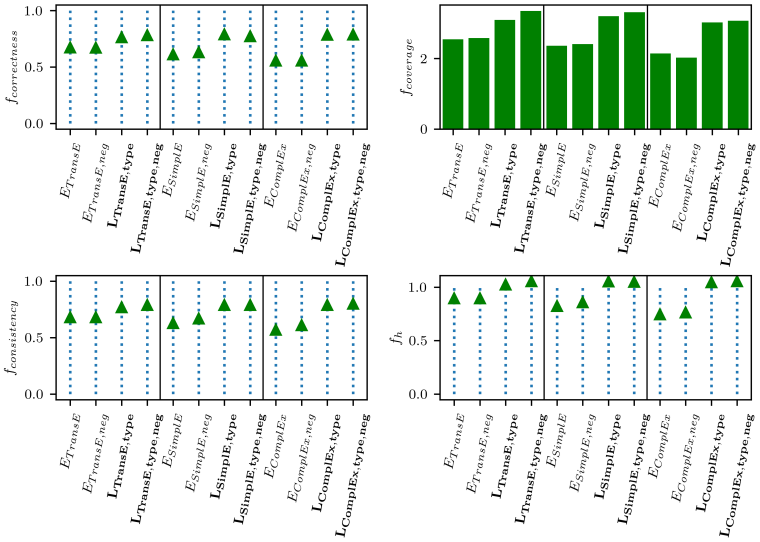

Fig. 7 NELL-995 dataset. Compare E-method, L-method and schema-aware negative sampling strategy in the schema-aware silver completeness performance

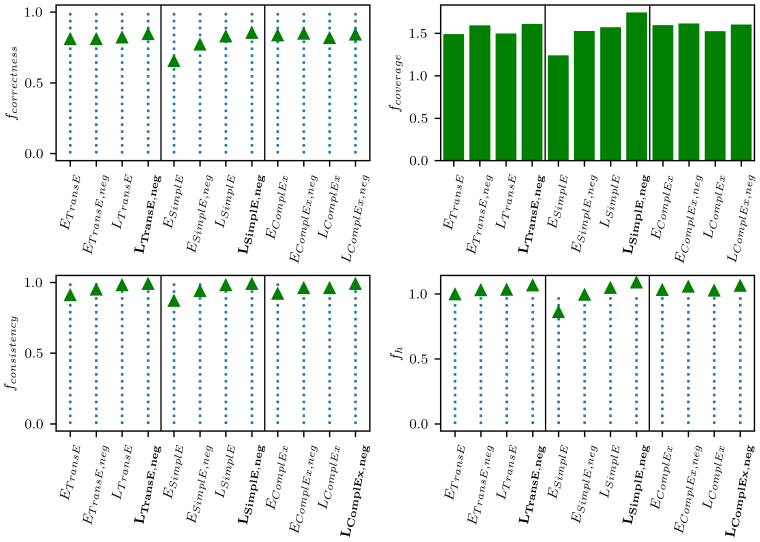

Fig. 8 DBped-P dataset. Compare E-method, L-method and schema-aware negative sampling strategy in the schema-aware silver completeness performance

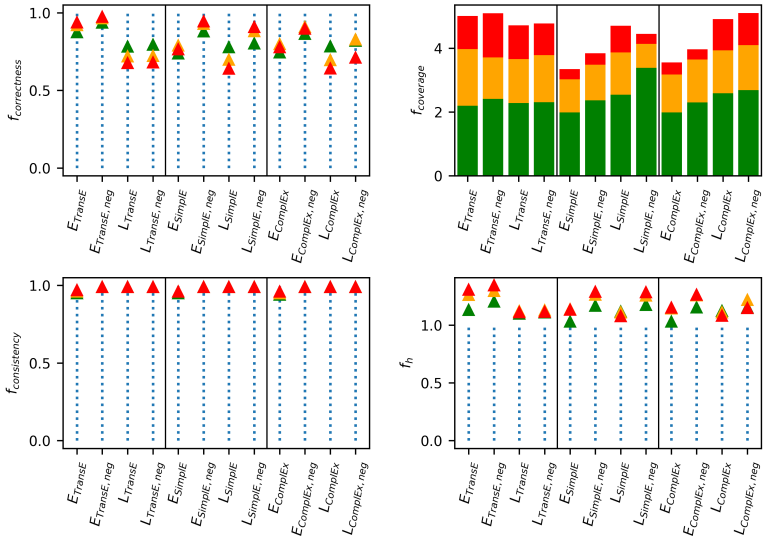

Fig. 9 TREAT dataset. Compare E-method, L-method and schema-aware negative sampling strategy in the schema-aware silver completeness performance

In Figures 10, 11 and 12, 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 10, the one round combinations outperformed most of the single methods running three rounds in $f_{coverage}$ with NELL-995 dataset. In Figure 11 and Figure 12, 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.

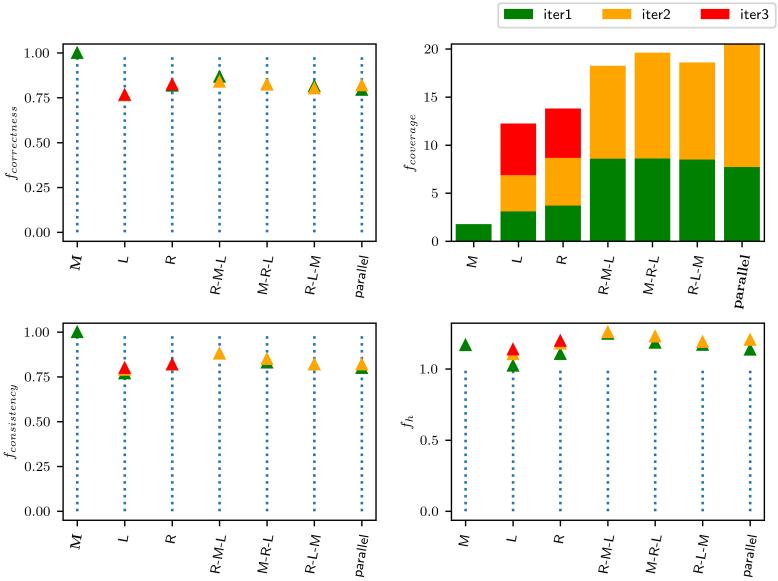

Fig. 10 NELL-995 dataset. Compare single methods and combined methods in the schema-aware silver completeness performance

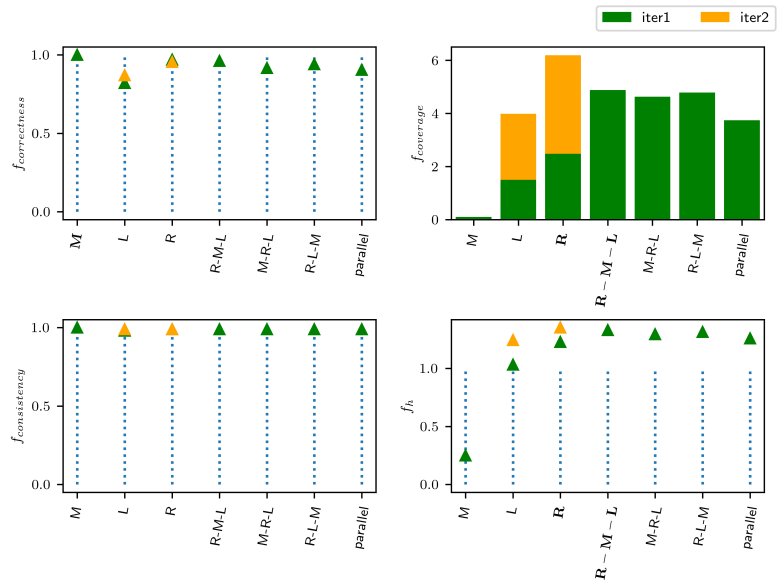

Fig. 11 DBped-P dataset. Compare single methods and combined methods in the schema-aware silver completeness performance

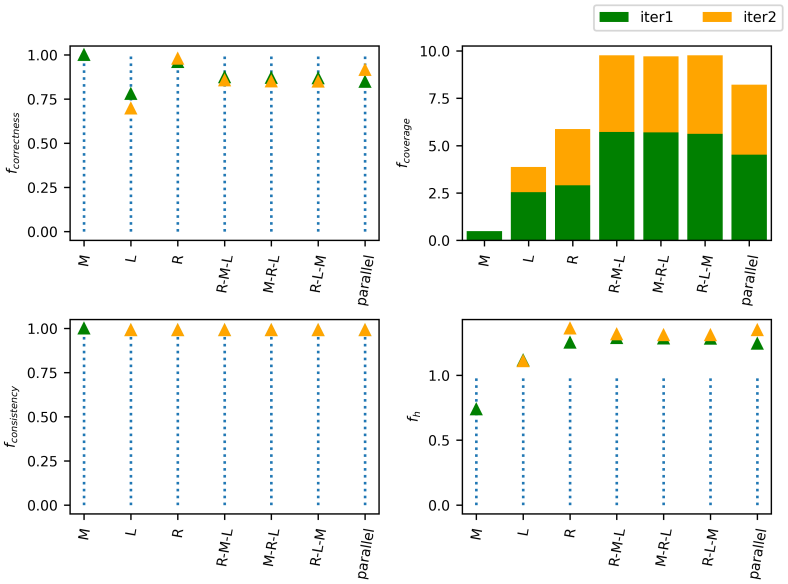

Fig. 12 TREAT dataset. Compare single methods and combined methods in the schema-aware silver completeness performance

We configured the embedding dimension for TransE, SimplE and ComplEx to 128 on all three datasets. The learning rate, batch size and maximum training iterations were configured w.r.t. the size of the training set and the loss convergence. 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 one iteration, while it increased to 2 hours in 3 iterations as the number of triples increased. 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.

5.4.2 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 hit@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 only, 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, 4, 5, and the results of TP are described in Tables 6, 7 and 8. 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.

Based on these results, we made the following analysis:

E-methods VS L-methods

As described in Table 3, 4 and 5, 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 [80].

Table 3 NELL dataset schema-aware silver standard evaluation in LP.

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

While as described in Table 6 and 7, the L-methods outperformed the E-methods in the TP task, given high quality literal and text encoders. In our experiment 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 entity literal can effectively improve LP and TP performance. However, literal does not bring the expected benefits to TREAT dataset in TP in the first iteration. A possible reason is that the text encoder for TREAT dataset is very different from the encoder that used for NELL-995 and DBped-P. The text encoder for TREAT literal was trained on technical manual and 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 similar in their surface form, but distinguished in context, which increases the challenge of training with literal encoder. This is another observation that literal qualities affect L-method's 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. While with the TREAT dataset, this sampling strategy decreased the performance in LP and TP

Table 4 DBped-P datasets schema-aware silver standard evaluation in LP.

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

in 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.

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 need 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 6, 7 and 8 confirmed this hypothesis. On 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 increase more than the E-methods, with the increase of iteration. We further

Table 5 TREAT datasets schema-aware silver standard evaluation in LP.

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

inject schema-correct data from R-method and M-method in KG embedding training, by running combined pipelines in iterative manner. In Table 9, 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.

5.5 Comparing our ACC with existing Reasoners

We evaluate our ACC approach with a designed use case: given a KG, we extend it with a set of random generated relation assertions, then we collect all the justifications that are generated by a tractable reasoner. We check whether our approach can detect all the incorrect relation assertions that are stated in each of these explanations. We compare our ACC approach with exiting reasoners in terms of:

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

In doing reasoning, the existing reasoner has 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

Table 6 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	-	-	-

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 [104], an approximate reasoner TrOWL and the ACC service provided by SIC. For both Hermit and TrOWL, we implement 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, NELL-995, and TREAT, but expand the initial KG by adding a few randomly generated inconsistent triples. The results of comparison can be seen in Table 10 In Table 10, *Exp* stands for the number of explanations that the justification service of HermiT generated. Each explanation consists of one or several axioms that cause 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 explanations detected by the HermiT's justification service are identified by

Table 7 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**	-	-

TrOWL, SIC, and our approach. *TT* stands for "TBox Transformation". Our TBox transformation is the most costly among 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.

6 Related Work

The work presented in this paper is most related to those studies addressing combining logical rules and KGE models for KGC. A few studies [105–108] aimed at learning joint models that inject logic into KG embeddings to obtain more predictive entity and relation embeddings. Approaches belonging to this category tightly integrate rule learning and embedding based approaches,

Table 8 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**

but are mostly restricted to a type of rule which does not allow for constants. EmbedS [109], TransC [60], Cose [110] and OntoZSL [111] 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 [105] learns KGEs by jointly modeling translation based embedding and logic, where logical rules are represented as first-order logic formulae and modelled by t-norm fuzzy logics. TRANSOWL [61] 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 [73], translation based methods cannot properly capture simple rules. Some joint models may be limited by the expressive capability of their base translation based models.

Some recent work tries to combine KGE and rule-learning in a collaborative or complementary manner. The IterE [90] was designed to address the sparsity problem for embedding learning and also the efficiency problem for

Table 9 The schema-aware silver standard evaluation in TP, compared the L-methods and the combination pipeline of R-method, M-method and L-methods. In the combination pipeline, the KG embeddings fed to TP are learned from last L-method in the series pipeline. We use the base embedding model having highest TP scores in the combination pipeline. For NELL-995, we used the type extended literals for L-method.

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	-
TREAT	1	Pr	0.86	0.88
		Rec	0.87	0.85
		F1	0.87	0.86
	3	Pr	0.93	**0.98**
		Rec	0.93	**0.96**
		F1	0.93	**0.97**

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 [94] constructs a simple method to combine the outcomes of rule-based and latent approaches in a post-processing step. Its combination strategy ensures that the rule-based method and the KG embedding model operate independently, but interact by aggregating rankings, for example, using the KG embedding 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. Our baseline SIC [72] iteratively exploits existing KGC methods and schema based logical reasoning, for both producing triples and checking schema correctness. Compared to SIC, our SICKLE has a series of new features, such as literal-based triple producer

Table 10 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
TREAT	55	704ms	0:59:22	100%	265ms	0:12:13	100%	31s	17s	100%	0:00:1	1s	85ms

and schema-aware negative sampling, which improves the schema-correctness. We also re-designed the ACC module to make it more complete.

Another strand of research has focused on applying logic rules in their KGE sampling strategies. In [107], a method was proposed to encode logical consistency into the distributed representation to make high-rank triples as consistent as possible. On one hand, triples entailed by datalog rule are selected as positive examples. On the other hand, it adds optimal objective function on certain negative triples that are inconsistent with the logical constraints. But it only used a small set of inconsistent triples in the optimal objective function for consideration of efficiency and scalability. TRANSOWL [61] avoids false positives by exploiting the available axioms, specified in RDFS and OWL, namely domain, range, disjointWith, functionalProperty, to generate negative examples. A more recent work[97] proposed a method that leverages schema to dynamically generate inconsistent triples as negative examples in its training procedure. Their consistency checking strategy assumes the given ontologies are in DL-Lite. It considered the syntax and semantics that can be directly translated from OWL to $DL-Lite^{S\sqcup}$, however it is not clear how it handles other syntax in OWL 2. These sampling strategies are similar to our schema-aware negative sampling strategy, in that the ontological knowledge is taken into account for the generation of negative samples. But our work is different from it in at least two ways. On the one hand, our approximate consistency checking method takes into account both efficiency and rich expressive power by approximate TBox from OWL 2 to DL-Lite. On the other hand, we apply schema-aware sampling strategy on iterative KGC pipeline with combinations of different KGC approaches and ACC module. 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.

7 Conclusion and Future Work

In this paper, 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 perform approximate consistency checking on produced triples, so that only schema-correct triples are added to the target KG.

Previous research [73] 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 schema

of Knowledge Graphs. We tackle this problem by considering a few strategies from different level. From system level, we combined different types of triple producers and an approximated consistency checking module to produce schema-correct triples and run it in iterative manner. Only schema-correct triples are added to the target KG and used in iterative training. From a functional level, we integrate a 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.

There are many potential paths for future work. Firstly, it would be interesting to include probabilistic reasoning to rank the new triples. The final rank could be based on a combination of scores from several different triple producers. Secondly, it might be an idea to further restrict our notion of correctness. Even though the quality of schema-correct triples is a lot better than the schema-incorrect ones, it does not mean that all the schema-correct triples are semantically correct. One straightforward approach is to further include some negative rules as constraints [68, 112], in addition to the domain and range constraints considered in this paper. Thirdly, it might be interesting to see how our proposed framework can be applied in some dynamic settings, such as reasoning and learning for stream Knowledge Graphs [113] and temporal Knowledge Graph completion [114–116]. Furthermore, we want to integrate and extend our framework with knowledge extraction, for example, unsupervised learning triples from pre-trained language models [117] and extracting triples from streamed data [67, 118].

Declarations

Ethical Approval

Not applicable.

Authors' contributions

Fangrong Wang did all the experiments, as well as some of the related detailed design, and most of the writing. Jeff Z. Pan proposed the original ideas, supervised the work and writing, as well as revising the paper. The other authors contributed through discussions of the paper drafts and comments on them as well as helping with polishing the paper.

Competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

The authors would like to thank Huawei for supporting the research on which this paper was based under grant CIENG4721/LSC.

Availability of data and materials

https://github.com/sig4kg/SIKGC

References

[1] Pan, J.Z., Vetere, G., Gomez-Perez, J.M., Wu, H. (eds.): Exploiting Linked Data and Knowledge Graphs for Large Organisations. Springer, ??? (2017)

[2] Pan, J.Z., Calvanese, D., Eiter, T., Horrocks, I., Kifer, M., Lin, F., Zhao, Y.: Reasoning Web: Logical Foundation of Knowledge Graph Construction and Querying Answering. Springer, ??? (2017)

[3] Guha, R., McCool, R., Miller, E.: Semantic search. In: WWW '03: Proceedings of the 12th International Conference on World Wide Web, pp. 700–709 (2003)

[4] 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)

[5] 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), pp. 2180–2189. NAACL-HLT, ??? (2019)

[6] Gu, Y., Zhou, T., Cheng, G., Li, Z., Pan, J.Z., Qu, Y.: Relevance Search over Schema-Rich Knowledge Graphs. In: Proc. of the 12th ACM International WSDM Conference (WSDM2019), pp. 114–122 (2019)

[7] 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, pp. 1835–1844 (2018)

[8] Wang, X., Wang, D., Xu, C., He, X., Cao, Y., Chua, T.-S.: Explainable reasoning over knowledge graphs for recommendation. In: Proceedings

of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 5329–5336 (2019)

[9] Tu, K., Cui, P., Wang, D., Zhang, Z., Zhou, J., Qi, Y., Zhu, W.: Conditional graph attention networks for distilling and refining knowledge graphs in recommendation. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 1834–1843 (2021)

[10] Wu, S., Sun, F., Zhang, W., Xie, X., Cui, B.: Graph neural networks in recommender systems: a survey. ACM Computing Surveys **55**(5), 1–37 (2022)

[11] 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), pp. 950–958 (2019)

[12] Xian, Y., Fu, Z., Muthukrishnan, S., de Melo, G., Zhang, Y.: Reinforcement knowledge graph reasoning for explainable recommendation. In: Proceedings of SIGIR, pp. 285–294 (2019)

[13] Yang, Y., Huang, C., Xia, L., Li, C.: Knowledge graph contrastive learning for recommendation. In: Proceedings of SIGIR, pp. 1434–1443 (2022)

[14] Wu, H., Wang, M., Zeng, Q., Chen, W., Nind, T., Jefferson, E.R., Bennie, M., Black, C., Pan, J.Z., Sudlow, C., Robertson, D.: Knowledge Driven Phenotyping. In: Proc. of Medical Informatics Europe (MIE 2020), pp. 1327–1328 (2020)

[15] Tripodi, I.J., Callahan, T.J., Westfall, J.T., Meitzer, N.S., Dowell, R.D., Hunter, L.E.: Applying knowledge-driven mechanistic inference to toxicogenomics. Toxicology in Vitro (2020)

[16] Zhang, R., Hristovski, D., Schutte, D., Kastrin, A., Fiszman, M., Kilicoglu, H.: Drug repurposing for covid-19 via knowledge graph completion. Journal of Biomedical Informatics **115** (2021)

[17] 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)

[18] Deng, S., Zhang, N., Zhang, W., Chen, J., Pan, J.Z., Chen, H.: Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network. In: Proc. of the World Wide Web Conference (WWW 2019), pp. 678–685 (2019)

[19] Cheng, D., Yang, F., Wang, X., Zhang, Y., Zhang, L.: Knowledge graph-based event embedding framework for financial quantitative investments. In: SIGIR, pp. 2221–2230 (2020)

[20] Zhu, X., Ao, X., Qin, Z., Chang, Y., Liu, Y., He, Q., Li, J.: Intelligent financial fraud detection practices in post-pandemic era. The Innovation **2** (2021)

[21] Xu, H., Giunchiglia, F.: Sko types: an entity-based scientific knowledge objects metadata schema. Journal of Knowledge Management **19**(1), 60–70 (2015)

[22] Auer, S., Kovtun, V., Prinz, M., Kasprzik, A., Stocker, M., Vidal, M.-E.: Towards a Knowledge Graph for Science. In: Proc. of the 8th International Conference on Web Intelligence, Mining and Semantics (WIMS 2018), pp. 1327–1328 (2018)

[23] Edelstein, E., Pan, J.Z., Soares, R., Wyner, A.: Knowledge-driven intelligent survey systems towards open science. New Generation Computing, 397–421 (2020)

[24] Pan, J.Z., Edelstein, E., Bansky, P., Wyner, A.: A Knowledge Graph Based Approach to Social Science Surveys. Data Intell. **3**(4), 477–506 (2021)

[25] Kelley, A., Garijo, D.: A framework for creating knowledge graphs of scientific software metadata. Quant. Sci. Stud. **2**, 1423–1446 (2021)

[26] 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**(2), 1–21 (2023)

[27] Xu, C., Guan, Z., Zhao, W., Wu, H., Niu, Y., Ling, B.: Adversarial incomplete multi-view clustering. In: IJCAI, vol. 7, pp. 3933–3939 (2019)

[28] Rospocher, M., van Erp, M., Vossen, P., Fokkens, A., Aldabe, I., Rigau, G., Soroa, A., Ploeger, T., Bogaard, T.: Building event-centric knowledge graphs from news. J. Web Semant. **37-38** (2016)

[29] Pan, J.Z., Pavlova, S., Li, C., Li, N., Li, Y., Liu, J.: Content based Fake News Detection Using Knowledge Graphs. In: Proc. of the International Semantic Web Conference (ISWC2018), pp. 669–683 (2018)

[30] 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)

[31] Liu, J., Wang, C., Li, C., Li, N., Deng, J., Pan, J.Z.: DTN: Deep triple network for topic specific fake news detection. J. Web Semant. **70** (2021)

[32] Phil, T., Z., P.J., Daniel, O., Evan, W., Michael, U., Elisa, K.: 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)

[33] Holger, K., Daniel, O., Phil, T., Evan, W., Z., P.J., Michael, U.: A semantic web primer for object-oriented software developers. W3C Working Group Note 9 March 2006, W3C (2006)

[34] Pan, J.Z., Staab, S., Aßmann, U., Ebert, J., Zhao, Y. (eds.): Ontology-Driven Software Development. Springer, ??? (2013)

[35] Pan, J.Z., Zhao, Y. (eds.): Semantic Web Enabled Software Engineering. IOS Press, ??? (2014)

[36] 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 (2021)

[37] Althar, R.R., Samanta, D.: The realist approach for evaluation of computational intelligence in software engineering. Innov. Syst. Softw. Eng. **17**, 17–27 (2021)

[38] 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), pp. 465–480 (2020)

[39] Buchgeher, G., Gabauer, D., Gil, J.M., Ehrlinger, L.: Knowledge graphs in manufacturing and production: A systematic literature review. IEEE Access **9** (2021)

[40] 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**(9), 1616–1637 (2018) arXiv:1709.07604. https://doi.org/10.1109/TKDE.2018.2807452

[41] Ji, S., Pan, S., Cambria, E., Marttinen, P., Yu, P.S.: A Survey on Knowledge Graphs: Representation, Acquisition and Applications, 1–26 (2020) arXiv:2002.00388

[42] Chen, X., Jia, S., Xiang, Y.: A review: Knowledge reasoning over knowledge graph. Expert Systems with Applications **141** (2020). https://doi.org/10.1016/j.eswa.2019.112948

[43] Hur, A., Janjua, N., Ahmed, M.: A Survey on State-of-the-art Techniques for Knowledge Graphs Construction and Challenges ahead (2021) arXiv:2110.08012

[44] 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**(2) (2021) arXiv:2002.00819. https://doi.org/10.1145/3424672

[45] Zhang, W., Chen, J., Li, J., Xu, Z., Pan, J.Z., Chen, H.: Knowledge Graph Reasoning with Logics and Embeddings: Survey and Perspective (2022) arXiv:2202.07412

[46] Xu, C., Zhao, W., Zhao, J., Guan, Z., Song, X., Li, J.: Uncertainty-aware multiview deep learning for internet of things applications. IEEE Transactions on Industrial Informatics **19**(2), 1456–1466 (2023). https://doi.org/10.1109/TII.2022.3206343

[47] A survey on deep learning based knowledge tracing. Knowledge-Based Systems **258**, 110036 (2022). https://doi.org/10.1016/j.knosys.2022.110036

[48] 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)

[49] 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 (2019) arXiv:1902.10197

[50] 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 (2017). https://doi.org/10.1007/978-3-319-63342-8_8

[51] 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 (2019). https://doi.org/10.24963/ijcai.2019/435

[52] 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 (2022). https://doi.org/10.1007/s11280-022-01110-6

[53] Yang, S., Cai, B., Cai, T., Song, X., Jiang, J., Li, B., Li, J.: Robust cross-network node classification via constrained graph mutual information. Knowledge-Based Systems **257**, 109852 (2022). https://doi.org/10.1016/j.knosys.2022.109852

[54] Fang, U., Li, J., Lu, X., Mian, A., Gu, Z.: Robust image clustering via context-aware contrastive graph learning. Pattern Recognition **138**, 109340 (2023). https://doi.org/10.1016/j.patcog.2023.109340

[55] Pan, W., Wei, W., Mao, X.-L.: Context-aware Entity Typing in Knowledge Graphs, 2240–2250 (2021) arXiv:2109.07990. https://doi.org/10.18653/v1/2021.findings-emnlp.193

[56] 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 (2022). https://doi.org/10.2139/ssrn.3985428

[57] Chen, J., Hu, P., Jimenez-Ruiz, E., Holter, O.M., Antonyrajah, D., Horrocks, I.: OWL2Vec*: embedding of OWL ontologies. Machine Learning **110**(7), 1813–1845 (2021) arXiv:2009.14654. https://doi.org/10.1007/s10994-021-05997-6

[58] Fionda, V., Pirrò, G.: Fact checking via evidence patterns. IJCAI International Joint Conference on Artificial Intelligence **2018-July**(ii), 3755–3761 (2018). https://doi.org/10.24963/ijcai.2018/522

[59] Fionda, V., Pirró, G.: Triple2Vec: Learning Triple Embeddings from Knowledge Graphs. Aaai (2019) arXiv:1905.11691

[60] 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 (2018) arXiv:1811.04588. https://doi.org/10.18653/v1/d18-1222

[61] 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 (2021). https://doi.org/10.1007/978-3-030-77385-4_26

[62] Bhatia, S., Dwivedi, P., Kaur, A.: Tell Me Why Is It So? Explaining Knowledge Graph Relationships by Finding Descriptive Support Passages (2018) arXiv:1803.06555

[63] Pirrò, G.: Fact-checking via path embedding and aggregation. CEUR

Workshop Proceedings **2722**, 149–158 (2020) arXiv:2011.08028

[64] Du, J., Pan, J.Z., Wang, S., Qi, K., Shen, Y., Deng, Y.: Validation of Growing Knowledge Graphs by Abductive Text Evidences. Proceedings of the AAAI Conference on Artificial Intelligence **33**, 2784–2791 (2019). https://doi.org/10.1609/aaai.v33i01.33012784

[65] 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). https://doi.org/10.1145/3289600.3290996

[66] 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 (2018). https://doi.org/10.1007/978-3-030-04284-4_2

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

[68] Ahmadi, N., Huynh, V.P., Meduri, V., Ortona, S., Papotti, P.: Mining Expressive Rules in Knowledge Graphs. Journal of Data and Information Quality **12**(2) (2020). https://doi.org/10.1145/3371315

[69] Loster, M., Mottin, D., Papotti, P., EhmA$\frac{1}{4}$ller, J., Feldmann, B., Naumann, F.: Few-shot knowledge validation using rules. The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021, 3314–3324 (2021). https://doi.org/10.1145/3442381.3450040

[70] Paulheim, H.: Knowledge graph refinement: A survey of approaches and evaluation methods. Semantic Web **8**(3), 489–508 (2016). https://doi.org/10.3233/SW-160218

[71] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E.R., Mitchell, T.M.: Toward an architecture for never-ending language learning. Proceedings of the National Conference on Artificial Intelligence **3**, 1306–1313 (2010)

[72] Wiharja, K., Pan, J.Z., Kollingbaum, M.J.: Schema aware iterative Knowledge Graph completion. Journal of Web Semantics, 100616 (2020). https://doi.org/10.1016/j.websem.2020.100616

[73] 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 (2018) arXiv:1805.10461

[74] Kazemi, S.M., Poole, D.: Simple embedding for link prediction in knowledge graphs. Advances in Neural Information Processing Systems **2018-Decem**(NeurIPS), 4284–4295 (2018) arXiv:1802.04868

[75] 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 (2016) arXiv:1606.06357

[76] 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)

[77] Shi, B., Weninger, T.: Open-world knowledge graph completion. 32nd AAAI Conference on Artificial Intelligence, AAAI 2018, 1957–1964 (2018) arXiv:1711.03438

[78] 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 (2017) arXiv:1604.04835

[79] Niu, L., Fu, C., Yang, Q., Li, Z., Chen, Z., Liu, Q., Zheng, K.: Open-world knowledge graph completion with multiple interaction attention. World Wide Web **24**(1), 419–439 (2021). https://doi.org/10.1007/s11280-020-00847-2

[80] Daza, D., Cochez, M., Groth, P.: Inductive Entity Representations from Text via Link Prediction. In: Proceedings of the Web Conference 2021, vol. 1, pp. 798–808. ACM, New York, NY, USA (2021). https://doi.org/10.1145/3442381.3450141. http://arxiv.org/abs/2010.03496%0Ahttp://dx.doi.org/10.1145/3442381.3450141 https://dl.acm.org/doi/10.1145/3442381.3450141

[81] Gesese, G.A., Biswas, R., Alam, M., Sack, H.: A Survey on Knowledge Graph Embeddings with Literals: Which model links better Literal-ly? **0**(0) (2019) arXiv:1910.12507

[82] Teru, K.K., Denis, E.G., Hamilton, W.L.: Inductive relation prediction by subgraph reasoning. 37th International Conference on Machine Learning, ICML 2020 **PartF16814**(1), 9390–9399 (2020) arXiv:1911.06962

[83] Selman, B., Kautz, H., Hill, M.: nowledge Compilation Horn Approximations (1991)

[84] Kautz, H., Selman, B.: Knowledge compilation and theory approximation. Journal of the ACM **43**, 193–224 (1996)

[85] 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)

[86] 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**(12), 2724–2743 (2017). https://doi.org/10.1109/TKDE.2017.2754499

[87] Nguyen, D.Q.: A survey of embedding models of entities and relationships for knowledge graph completion, 1–14 (2017) arXiv:1703.08098

[88] Chen, Z., Wang, Y., Zhao, B., Cheng, J., Zhao, X., Duan, Z.: Knowledge graph completion: A review. IEEE Access **8**, 192435–192456 (2020). https://doi.org/10.1109/ACCESS.2020.3030076

[89] Meilicke, C., Fink, M., Wang, Y., Ruffinelli, D., Gemulla, R., Stuckenschmidt, H.: 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 (2018). https://doi.org/10.1007/978-3-030-00671-6_1

[90] Zhang, W., Chen, J., Paudel, B., Zhu, H., Wang, L., Zhang, W., Bernstein, A., Chen, H.: Iteratively learning embeddings and rules for knowledge graph reasoning. The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019, 2366–2377 (2019) arXiv:1903.08948. https://doi.org/10.1145/3308558.3313612

[91] 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 (2020). https://doi.org/10.1007/978-3-030-49461-2_3

[92] 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 (2019) arXiv:arXiv:2004.04412v1. https://doi.org/10.24963/ijcai.2019/435

[93] Pan, J.Z., Ren, Y., Zhao, Y.: Tractable approximate deduction for OWL. Artificial Intelligence **235**, 95–155 (2016). https://doi.org/10.1016/j.artint.2015.10.004

[94] 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)

[95] Qian, J., Li, G., Atkinson, K., Yue, Y.: Understanding Negative Sampling in Knowledge Graph Embedding. International Journal of Artificial Intelligence & Applications **12**(1), 71–81 (2021). https://doi.org/10.5121/ijaia.2021.12105

[96] Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. Proceedings of the National Conference on Artificial Intelligence **2**(June 2014), 1112–1119 (2014). https://doi.org/10.1609/aaai.v28i1.8870

[97] 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 (2021). https://doi.org/10.1007/978-3-030-88361-4_24

[98] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv, 4171–4186 (2018)

[99] Shearer, R., Motik, B., Horrocks, I.: HermiT: A highly-eficient OWL reasoner. CEUR Workshop Proceedings **432** (2009)

[100] 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**(PART 2), 431–435 (2010). https://doi.org/10.1007/978-3-642-13489-0_38

[101] Steigmiller, A., Liebig, T., Glimm, B.: Konclude : System Description (2014)

[102] Pan, J.Z., Vetere, G., Gomez-Perez, J.M., Wu, H.: Exploiting Linked Data and Knowledge Graphs in Large Organisations. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-45654-6. http://link.springer.com/10.1007/978-3-319-45654-6

[103] Pan, J.Z., Thomas, E.: Approximating OWL-DL Ontologies. AAAI, 1434–1439 (2007)

[104] Shearer, R., Motik, B., Horrocks, I.: HermiT: A Highly-Efficient Reasoner for Description Logics. Cs.Ox.Ac.Uk, 1–13 (2008)

[105] 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). https://doi.org/10.18653/v1/d16-1019

[106] Guo, S., Ding, B., Wang, Q., Wang, L., Wang, B.: Knowledge Base Completion via Rule-Enhanced Relational Learning. In: Communications in Computer and Information Science vol. 650, pp. 219–227 (2016). https://doi.org/10.1007/978-981-10-3168-7_22. http://link.springer.com/10.1007/978-981-10-3168-7_22

[107] 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 (2018). https://doi.org/10.1007/978-3-030-01716-3_11

[108] 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**(ii), 110–121 (2018) arXiv:1805.02408. https://doi.org/10.18653/v1/p18-1011

[109] Diaz, G., Fokoue, A., Sadoghi, M.: EmbedS: Scalable, Ontology-aware Graph Embeddings. (2018). https://doi.org/10.5441/002/edbt.2018.40

[110] Gao, H., Zheng, X., Li, W., Qi, G., Wang, M.: Cosine-Based Embedding for Completing Schematic Knowledge. In: Tang, J., Kan, M.-Y., Zhao, D., Li, S., Zan, H. (eds.) Natural Language Processing and Chinese Computing, pp. 249–261. Springer, Cham (2019)

[111] Geng, Y., Chen, J., Chen, Z., Pan, J.Z., Ye, Z., Yuan, Z., Jia, Y., Chen, H.: OntoZSL: Ontology-enhanced zero-shot learning. In: The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021, pp. 3325–3336. ACM, New York, NY, USA (2021). https://doi.org/10.1145/3442381.3450042. https://dl.acm.org/doi/10.1145/3442381.3450042

[112] Galárraga, L.A., Teflioudi, C., Hose, K., Suchanek, F.: AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases, 413–422 (2013). https://doi.org/10.1145/2488388.2488425

[113] Kazemi, S.M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., Poupart, P.: Representation learning for dynamic graphs: A survey.

Journal of Machine Learning Research **21**, 1–73 (2020) arXiv:1905.11485

[114] 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). https://doi.org/10.18653/v1/d18-1225

[115] Lacroix, T., Obozinski, G., Usunier, N.: Tensor Decompositions for temporal knowledge base completion, 1–12 (2020) arXiv:2004.04926

[116] Liu, Y., Hua, W., Xin, K., Zhou, X.: Context-Aware Temporal Knowledge Graph Embedding. Lecture Notes in Computer Science, vol. 11881, pp. 583–598. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34223-4_37. http://link.springer.com/10.1007/978-3-030-34223-4{_}37

[117] Wang, C., Liu, X., Song, D.: Language Models are Open Knowledge Graphs, 1–30 (2020) arXiv:2010.11967

[118] 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 (2021). https://doi.org/10.1007/978-3-030-77385-4_38

A Results in Table

The data in Table 12, 11, and 13 correspond to figures in section 5.

Table 11 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

Table 12 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
E_{SimplE}	2	1,157,494	0.62	4.56	0.64	0.89
$\text{E}_{\text{ComplEx}}$	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
$\text{L}_{\text{ComplEx}}$	2	1,174,789	0.76	4.65	0.59	0.80
$\text{L}_{\text{TransE,type}}$	2	1,635,268	0.67	6.86	0.79	1.10
$\text{L}_{\text{SimplE,type}}$	2	1,652,168	0.78	6.94	0.73	1.00
$\text{L}_{\text{ComplEx,type}}$	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
$\text{Base}_{\text{TransE}}$	3	2,162,641	0.33	9.39	0.16	0.32
$\text{Base}_{\text{SimplE}}$	3	1,490,708	0.31	6.16	0.12	0.25
$\text{Base}_{\text{ComplEx}}$	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
$\text{E}_{\text{ComplEx}}$	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
$\text{L}_{\text{ComplEx}}$	3	1,955,428	0.60	8.40	0.61	0.88
$\text{L}_{\text{TransE,type}}$	3	2,936,882	0.76	12.23	0.80	1.14
$\text{L}_{\text{SimplE,type}}$	3	2,356,820	0.71	10.33	0.72	1.02
$\text{L}_{\text{ComplEx,type}}$	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

Table 13 TREAT 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	47,266	1.0	0.48	1.0	0.65
R	1	159,296	0.70	2.43	0.99	1.44
$\text{Base}_{\text{TransE}}$	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
E_{SimplE}	3	173,047	0.76	3.34	0.96	1.13
$\text{E}_{\text{ComplEx}}$	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
$\text{L}_{\text{ComplEx}}$	3	235,414	0.64	4.91	0.99	1.08
$\text{E}_{\text{TransE,neg}}$	3	242,761	0.97	5.10	0.99	1.34
$\text{E}_{\text{SimplE,neg}}$	3	192,639	0.94	3.84	0.99	1.29
$\text{E}_{\text{ComplEx,neg}}$	3	197,555	0.90	3.96	0.99	1.26
$\text{L}_{\text{TransE,neg}}$	3	229,777	0.68	4.77	0.99	1.11
$\text{L}_{\text{SimplE,neg}}$	3	216,870	0.91	4.44	0.99	1.28
$\text{L}_{\text{ComplEx,neg}}$	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