Semantic Web 0 (0) 1
IOS Press


# Automatic Detection of Relation Assertion Errors and Induction of Relation Constraints

Andre Melo [a] and Heiko Paulheim [a]
[a] *Data and Web Science Group, University of Mannheim, Germany*
*E-mails: andre@informatik.uni-mannheim.de, heiko@informatik.uni-mannheim.de*

**Abstract.** Although the link prediction problem, where missing relation assertions are predicted, has been widely researched, error detection did not receive as much attention. In this paper, we investigate the problem of error detection in relation assertions of knowledge graphs, and we propose an error detection method which relies on path and type features used by a classifier for every relation in the graph exploiting local feature selection. Furthermore, we propose an approach for automatically correcting detected errors originated from confusions between entities. Moreover, we present an approach that translates decision trees trained for relation assertion error detection into SHACL-SPARQL relation constraints. We perform an extensive evaluation on a variety of datasets comparing our error detection approach with state-of-the-art error detection and knowledge completion methods, backed by a manual evaluation on DBpedia and NELL. We evaluate our error correction approach results on DBpedia and NELL and show that the relation constraint induction approach benefits from the higher expressiveness of SHACL and can detect errors which could not be found by automatically learned OWL constraints.

Keywords: shacl, sparql, ontology learning, error detection, knowledge graph

## 1. Introduction

Many of the knowledge graphs published as Linked Open Data have been created from semi-structured or unstructured sources. The sheer size of many of such knowledge graphs, e.g.: DBpedia, NELL, Wikidata, YAGO, do not allow for manual curation, and, instead, require the use of heuristics. Such heuristics, in turn, allow for the automatic or semi-automatic creation of large-scale knowledge graphs, but do not guarantee that the resulting knowledge graphs are free from errors. In addition, Wikipedia, which serves as source for DBpedia and YAGO, is estimated to have 2.8% of its statements wrong [70], which add up to the error caused by the extraction heuristics. Therefore, automatic approaches to detect wrong statements are an important tool for the improvement of knowledge graph quality.

Incompleteness is another major problem of most knowledge graphs. Automatic knowledge graph completion has been widely researched [46], with a variety of methods proposed, including embedding models. Although such methods can also be trivially employed for error detection, their performance has not yet been extensively evaluated on the task.

Many existing large-scale error detection methods rely exclusively on the types of subject and object of a relation [13, 52, 53], and try to spot violations of the underlying ontology and/or typical usage patterns. In the example depicted in Fig. 1, the error `president(Colin Powell, Bush (Illinois))` could be identified, since the entity `Bush (Illinois)` is of type city, but the relation `president` does not allow for cities in the object position (either by an explicit restriction in the ontology or by less formal conventions).

While types can be a valuable feature, some knowledge graphs lack this kind of information, have only incomplete type information, or have types which are not very informative. Moreover, some errors might contain wrong instances of correct types. For example, the

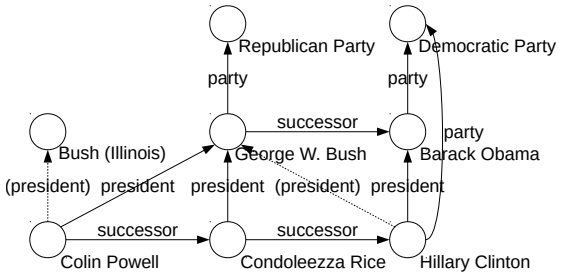



Fig. 1. Example excerpt from DBpedia. Two erroneous *president* relations, indicated by brackets and dashed lines, have been incidentally added.

fact `president(Hillary Clinton, George W. Bush)`, which is wrong, could not be detected with such an approach, because the schema is not violated: the `president` relation in this example expects persons both in the subject and object position, which is respected in this example.

In knowledge graph completion, paths in the graph have been proven to be valuable features [18, 27]. In the example depicted in Fig. 1, to predict whether a person $a$ is member of a party $b$ (`party`($a$,$b$)), one important feature is whether the president $a$ serves for is a member of party $b$, i.e., `president`($a$,$X$) $\rightarrow$ `party`($X$,$b$). Generalizing for any pair of entities in a given relation, we can consider `president` $\rightarrow$ `party` as a binary path feature to predict new edges in the knowledge graph. Typically, in knowledge graph completion, such paths are then exploited to predict missing relation assertions [17, 36]. For error detection, these features can complement the type features. However, searching for interesting paths for all the relations in a knowledge graph can be a challenging task, especially in datasets with many relations.

Once erroneous triples in a knowledge graph are detected, there are various ways of how to proceed. The simplest approach is to delete them, however, in some cases the erroneous relation assertions can be corrected instead. One common source of errors is the confusion between instances of a similar names [49, 53], as in the (artificial) example in Fig. 1, where `George W. Bush` was confused with `Bush (Illinois)`.[1]

By exploiting such cases, it is possible to also reduce incompleteness while reducing noise. This also helps reduce the search space of possible facts a knowledge graph could be enriched with. The number of possible relation assertions grows quadratically with the number of instances $n_c = n_i^2 n_r - n_f$, where $n_i$ is the number of instances, $n_r$ the number of relations and $n_f$ the number of existing facts in the graph. For large datasets such as DBpedia, Wikidata and YAGO, computing the confidence score of all these facts is challenging. While pruning possible facts which violate ontology constraints, especially domain and range restrictions of relations, can significantly reduce the search space, the problem is still very challenging. To illustrate the size of the search space, in DBpedia (2016-10) $n_c \approx 4.4 \times 10^{17}$ facts; when filtering those triples which violate the domain and range restriction the number is reduced to $n_c \approx 2.8 \times 10^{17}$.

When correcting wrong facts originated from confusions between entities, the search space is composed by the entities which could have been confused with the subject and the object. In many cases, the source of such confusions are entities with the same or similar names. Hence, in order to find candidates entities, we can e.g., exploit Wikipedia disambiguation links (which identifies entities which are often confused with each other), or use approximate string matching.

Another interesting field of research is the derivation of higher level patterns from the errors found in a knowledge graph. There are two major motivations: (1) for validating the results of error detection, a user can inspect a small number of patterns instead of a large number of individual mistakes [53]. Furthermore, given that errors follow typical patterns, (2) a set of higher level patterns can be directly deployed in the knowledge graph creation process, or even for live updates.

The problem of finding higher level patterns is addressed by ontology induction approaches, which normally represent relation constraints in the form of RDFS domain and range restrictions. Since designing a good ontology can be a challenging task, there has been a lot of work on learning ontologies from data, using methods such as inductive logic programming (ILP) [8] or association rule mining [66] for automatically learning ontology axioms.

One of the main problems with these methods is the restricted expressiveness of the learned ontologies. Modern knowledge graphs are often complex, and constraints may require the use of more expressive axioms which cannot be learned by current state-of-the-art methods. Furthermore, the intended and the actual use of a property often diverge, leading to situ-

[1] An actual example from DBpedia is the fact `formerTeam(Alan_Ricard, Buffalo_Bill)`, which originates from an error in Wikipedia: instead of referring to the NFL team `Buffalo_Bills`, the link in Wikipedia was erroneously pointing to the person `Buffalo_Bill`.

ations where a single ontology can hardly describe the different, often competing usages of a property.

One example for the latter case is the `president` in DBpedia. The relation is originally conceived to be used to define the person who presides an organization, hence in DBpedia's ontology it has the domain `Organisation` and range `Person`. However, the relation is also frequently used to define the president which a member of the government served, as in Figure 1. In order to allow the co-existence of both usages, the domain of the relation should be more flexible accepting `Organisation` or `Person`. One possible solution using RDFS domain and range axioms is to use the most specific common parent of the two classes, that is `Agent`, however, that also allows subjects to be of the classes `Deity`, `Family`, which would be undesirable. Another possible solution is to specify the union of `Organisation` and `Person` as the domain or range of a relation, however, using such disjunctions can drastically increase the reasoning complexity [21], which can be a major design factor for the implementation on large-scale knowlegde graphs [54], in particular in live settings.

In the example above, path constraints can be useful for describing the relation. In DBpedia, both members of the government and presidents have `successor` relation assertions indicating the person who occupied their respective positions after them. We know that a member of the government should have the same president as its predecessor, or the successor of the president of its predecessor. The former case happens when a president has, e.g., different secretary of states during its government, and the latter when the secretary of state is the first nominated by a new president. This can be represented with the disjunction of two graph path constraints:[2]

$$\texttt{president}(a,b) \rightarrow \quad (\texttt{successor}(c,a) \wedge \texttt{president}(c,b)) \vee (\texttt{successor}(c,a) \wedge \texttt{president}(c,d) \wedge \texttt{successor}(d,b))$$

We assume that each variable only occurs once in a path, i.e., the underlying patterns are acyclic. However, as in the example above, constraints may be formulated using multiple *paths*, which allows also for validating patterns of that kind.

[2]While disjunction can be problematic for the complexity of general purpose OWL reasoners, data validation with disjunctive patterns can be performed rather efficiently.

With the method we propose in this paper, we are able to learn such complex logical expressions, which subsume path patterns and simple domain and range restrictions. The patterns are expressed in the language SHACL, which is particularly designed for data validation.[3]

This paper addresses the following research questions:

**RQ1:** *How can we efficiently detect wrong assertion errors?*

We propose a hybrid approach called *PaTyBRED* (Paths and Types with Binary Relevance for Error Detection), which incorporates type and path features into local relation classifiers which predict whether a pair of subject and object belongs to a relation or not.

**RQ2:** *How can we describe the error detection process and integrate it into the knowledge graph?*

We propose a method for translating a PaTyBRED model learned with decision trees as classifiers into SHACL relation constraints. SHACL[4] (Shapes Contraint Language) is a versatile constraints language for validating RDF graphs, with which we are able to generate expressive and flexible relation constraints and better handle incomplete and noisy datasets.

**RQ3:** *How can we automatically correct some errors originated from confusions between entities?*

We propose CoCKG, an automatic correction approach which identifies and resolves relation assertion errors caused by confusion between instances. The approach relies on error detection methods as well as type predictors to assess the confidence of the corrected facts. It uses approximate string matching and exploits both searching for entities with similar IRIs as well as Wikipedia disambiguation pages (if available) to find candidate instances for correcting the facts.

This paper is an extension of [38], which addresses the detection of relation assertion errors problem, and [37], which introduces the idea of correction of confusions between entities. As part of the extension, we propose the learning of SHACL relation constraints, and perform evaluations on additional knowledge graphs.

[3]https://www.w3.org/TR/shacl/

[4]https://www.w3.org/TR/shacl/

In the experiments, we perform an extensive comparison of our PaTyBRED with state-of-the-art error detection and knowledge completion methods, and we conduct a manual evaluation of our approach on DBpedia and NELL, as well as evaluate the scalability using synthetic knowledge graphs. Furthermore, we manually evaluate the suggestions made by CoCKG, and we evaluate the generated SHACL relation constraints and perform another manual evaluation comparing them with domain and range restrictions induced with Statistical Schema Induction [66].

## 2. Problem Definition

We define a knowledge graph $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the T-box and $\mathcal{A}$ is the A-box containing relations assertions $\mathcal{A}_R$, type assertions $\mathcal{A}_C$, and literal assertions $\mathcal{A}_L$, where the latter are mentioned for the sake fo completeness, but not further considered in this course of this work. We define $N_C$ as the set of concepts (types), $N_R$ as the set of relations and $N_I$ as the set of individuals (entities which occur as subject or object in relations). The set of relation assertions is defined as $\mathcal{A}_R = \{r(s,o) | r \in N_R \wedge s, o \in N_I\}$ and the set of type assertions as $\mathcal{A}_C = \{C(s) | C \in N_C \wedge s \in N_I\}$. It is important to note that on RDF data $\mathcal{A}_R$ corresponds to links between entities (i.e., `owl:ObjectProperty`), and $\mathcal{A}_C$ corresponds to `rdf:type` assertions.

The problem addressed by research question RQ1 is the detection of erroneous relation assertions in the set $\mathcal{A}_R$. In practice, an approach for erroneous relation assertion detection is given a knowledge graph containing errors, and creates a function $\mathcal{A}_R \rightarrow [0,1]$, which assigns a score to the model. Using those scores, we reformulate the error detection as a ranking problem, i.e., erroneous relations should be ranked consistently higher than correct ones. In order to make the approach as versatile and applicable to as many knowledge graphs as possible, we do not use any other information, such as textual or numerical literals, or external knowledge sources. The problem can be defined as relation assertions error detection on internal features according to [50].

The problem addressed by research question RQ2 is the induction of relation constraints from data. That is, instead of trying to directly improve $\mathcal{A}_R$, the objective is to learn relation constraints in order to extend the T-box $\mathcal{T}$. A better quality T-box might be able to more effectively detect inconsistencies in the A-box, indirectly improving it at as a consequence, at the same time providing reusable and human interpretable artifacts as a result.

The problem addressed by research question RQ3 is the identification and correction of errors generated by confusions between entities. In this paper, we assume that errors originate from a confusion in either the subject or object entity. That is, an originally correct relation assertion $r(s,o) \notin \mathcal{A}_R$ is not only missing in the knowledge graph, but represented as an incorrect fact $r(s,o')$ or $(s',o)$, such that $s' \neq s$, $o' \neq o$, and $s, o, s', o' \in N_I$. The goal is to identify such cases and find the originally correct $r(s,o)$ given the corrupted triple $r(s,o')$ or $r(s',o)$ and the A-box, i.e. $\mathcal{A}_R$ and $\mathcal{A}_C$.

## 3. Related Work

The works related to this paper can be divided into two parts: detection and correction of relation assertion errors (related to RQ1 and RQ3), which includes error detection and knowledge completion models, and ontology learning, which includes works which induce ontology axioms from data, more specifically relation constraints (related to RQ2). In the next subsections we discuss each part in more details.

### *3.1. Detection of Relation Assertion Errors*

The problem of relation assertion error detection in knowledge graphs has been intensively researched by the Semantic Web community. As discussed in the introduction, there are erroneous relation assertions that are at the same time a violation to the ontology or T-Box of the knowledge graph (e.g., referring to a city instead of a sports team), while others are not (e.g., referring to one person instead of another). Apart from synonyms, a lack of domain and range restrictions of relations or too general restrictions is one of the main causes of problems of the latter category. Most recent methods proposed for cleansing large-scale LOD knowledge graphs, such as DBpedia and NELL, therefore do not rely solely on the schema, but use characteristics of the knowledge graph's A-box to detect erroneous assertions. A detailed survey including link prediction and error detection methods for knowledge graphs can be found in [50].

SDValidate [52] exploits statistical distributions of types and relations, and [13] applies outlier detection on type-based entity similarity measures to detect erroneous relation assertions. In more detail, SDValidate computes a distribution of object types for a given

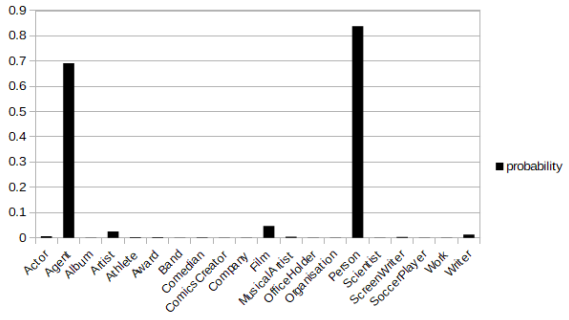



Fig. 2. Example distribution of the object types of the DBpedia property `director`

property. For a given relation assertion `x p o`, SDValidate compares all the types of `o` to the distribution of `p`, and computes a confidence score based on the deviation of those types from the distribution. An example for such a distribution is shown in figure 2. A relation assertion with property `director` and an object which has `Agent` and `Person` as types would receive a high confidence, whereas an assertion with an object of a different types (e.g., a company, for example a movie studio) would receive a lower score. These methods can effectively detect errors on DBpedia, however they require the existence of informative type assertions. Moreover, more complex errors containing wrong entities with correct types cannot be detected.

Knowledge graph completion (KGC) is a field highly related to error detection. Despite addressing a different problem, many KGC methods can also be used on the problem addressed in this paper. This kind of methods can be divided into graph-based, which relies on features which can be directly observed in the graph, and embedding methods, which learn latent features that represent entities and relations in an embedding space.

The Path Ranking Algorithm (PRA) [27] has shown that a logistic regression classifier using path features generated with random walks can be used for learning and inference in KGs and outperforms N-FOIL horn-clause inference on NELL [28]. PRA learns HORN clauses to predict relations, e.g., $citizenOf(X,Y) \leftarrow livesIn(X,Z), country(Z,Y)$. To scale to large knowledge graphs, random walks are used to generate the path features instead of attempting to fully enumerate the search space.

In later works, the PRA approach has been improved with Sub-graph Feature Extraction (SFE) [18], which also simplifies aspects of PRA. For instance, while PRA uses real valued features which correspond to the probabilities to reach $o$ from $s$ with a given path, SFE simply uses binary features which indicate if $o$ can be reached from $s$ or not. SFE not only reduces runtime by an order of magnitude when compared with PRA, but it also improves the qualitative performance.

In the recent years, knowledge graph embedding models, i.e., projections of knowledge graphs into lower-dimensional, dense vector spaces, have received a lot of attention [68]. Several different models have been developed for the knowledge graph completion problem and have brought improvements in performance.

There is a plethora of different embeddings models for knowledge graphs. One of the earliest embedding models is RESCAL [48], which performs tensor factorization on the knowledge graph's adjacency tensor, with the resulting eigenvectors corresponding to the entity embeddings and the core tensor the relations matrices. TRESCAL [10] extends RESCAL by exploiting entity types as well as domain and range restrictions of relations to improve the data quality and speed up the tensor factorization process. Neural Tensor Model (NTN) [62] represents each relation as a bilinear tensor operator followed by a linear matrix operator. Other early embedding models include Structure Embeddings (SE) [5], Semantic Matching Energy (SME) [3] and Latent Factor Model (LFM) [25].

Translation-based embeddings represent relations as translations between subject and object entities. TransE [4] was the first translation-based model and entities and relations share the same embeddings space. In TransH [69] and TransR [33] the translations are performed in the relations space, which is different from the entities space, and require projection matrices to map the entities onto the relations space. TransG [71] and CTransR [33] incorporate multiple relation semantics, where a relation may have multiple meanings determined by the entities pair associated with the relation. PTransE [32] extends TransE by considering relation paths as regular relations, which makes the number of relations considered grow exponentially.

Other approaches include DistMult [72], which uses dot product instead of translations to compute the triple scores. HolE [47] used circular correlation as an operator to combine the subject and object embeddings, Complex Embeddings [65] represents a triple score as the hermitian dot product of the relation, subject and object embeddings, which consist of real and imaginary vector components. ProjE [61] formulates the

knowledge graph completion as a ranking problem, and it optimizes the ranking of candidate entities collectively. It is reportedly one of the best performing KGC methods.

Some embedding models, such as RDF2Vec [58, 59] and Global RDF vectors [11], are not conceived for the KGC task and cannot generate triple scores. Thus, they cannot be directly used for error detection in the same way the other models mentioned earlier can, but in principle, they can serve as feature generation mechanisms for training relation scoring models.

Recently some works have raised doubts about the performance of new KGC embeddings models. Most of the experiments rely exclusively on two datasets (WN18 and FB15k), which contain many inverse relations [64]. Therefore some of the models may exploit this characteristic and not necessarily perform as well on other KGs. It has also been shown that the presence of relations between candidate pairs can be an extremely strong signal in some cases [64]. Moreover, recent works showed that a hyperparameter tuning has been overlooked and that a simple method, such as DistMult, can achieve state-of-the-art performance when well tuned [26].

### *3.2. Correction of Detected Errors*

As mentioned in the previous subsection, there are several different approaches for link prediction and some for error detection. It is important to note that none of those approaches mentioned address the problem of covering the candidate triples space (of size $n_c$ as discussed in the introduction). Our approach, on the other hand, exploits the assumption that erroneous facts often have a corresponding correct fact in order to reduce that space. Error detection approaches, such as SDValidate and PaTyBRED, focus on the detecting of already existing erroneous triples. It has been shown that state-of-the-art embeddings perform worse than PaTyBRED in the error detection task [38].

Rule-based systems, such as AMIE [17], cannot assign scores to arbitrary triples. However, they could be used to restrict the $n_c$ search space by identifying high confidence soft rules and using the missing facts from instances where the rule does not hold as candidates. Combining them with previously mentioned KG models would be an interesting line of reasearch, however, it is out of the scope of this paper.

Wang et al. [67] studied the problem of erroneous links in Wikipedia, which is also the source of many errors of DBpedia. They model the Wikipedia links as a weighted directed mono-relational graph, and propose the LinkRank algorithm which similar to PageRank, but instead of ranking the nodes (entities), it ranks the links. They use LinkRank to generate candidates for the link correction and use textual features from the description of articles to learn a SVM classifier that can detect errors and choose the best candidate for correction. While this is a closely related problem, which can help mitigate the problem studied in this paper, their method cannot be directly applied on arbitrary knowledge graphs. Our approach takes advantage of the multi-relational nature of KGs, entity types, ontological information and the graph structure.

### *3.3. Ontology Learning*

As discussed above, most works on detecting errors in knowledge graphs address the level of individual assertions, with the already mentioned shortcomings. There are few works which attempt to derive reusable, higher-level artifacts.

One such approach has been proposed in [63]. The authors provide means of learning additional domain and range restrictions for relations, which can then facilitate more fine-grained fact checking. The domain and range axioms learned are a reusable artifact, but, as discussed above, are not always suitable for the complex scenarios induced by modern knowledge graphs.

In [53], we have introduced an approach that clusters similar relation assertion errors. Those clusters can be more easily inspected by experts (e.g., by presenting them one typical, prominent example as a proxy for a class of errors), but the expert still needs to identify the cause and come up with a suitable fix manually.

The work presented in [49] aims at closing that gap by precisely pinpointing the cause of an error. For DBpedia, it is able to identify single axioms in the ontology or single mapping elements (i.e., the smallest building blocks of the creation process) that are responsible for a class of errors. It is, however, tightly tangled to the DBpedia creation process and cannot be trivially transferred to other knowledge graphs built with different methods.

Since we discuss the learning of constraints to be used for validating a knowledge graph, we target a problem which is similar to that of ontology learning or enrichment; a field in which quite a bit of related work exists. Rudolph [60] uses a class of OWL axioms that generalize domain and range restrictions, which support the conjunction of concepts. Statistical schema induction (SSI) [66] uses associa-

tion rule mining to learn OWL 2 EL axioms, such as class and relation subsumptions, relation's domain and range restrictions, relation transitiveness. Bühmann and Lehmann [8] propose a method for enriching ontologies with OWL 2 axioms implemented in the DL-Learner framework. Regarding relation assertion constraints, domain and range restrictions relation cardinalities [44] are the only kind of constraint which can be learned by these methods. A brief introduction to ontology learning and overview of the main approaches can be found in [31].

Gayo et al. [20] use SHACL and ShEX to define constraints to validate and describe linked data portals. Arndt et al. [1] uses rule mining to learn RDFCV (RDF Constraints Vocabulary). Swift Linked Data Miner (SLDM) [55] is the only system at the moment which can automatically learn SHACL constraints. However, it does not learn relation constraints, only class expressions.

Rule learning approaches, such as AMIE [17] and DL-Learner [30], could in principle have some of their rules converted into SHACL constraints. Since they were not originally conceived for learning relation constraints, these approaches would need to be extended in order to support it. As of now there are no works in that direction.

## 4. Detection of Relation Assertion Errors

In this section, we describe PaTyBRED (Paths and Types with Binary Relevance for Error Detection), a method for detecting relation assertion errors which relies both on path and type features. This method addresses research question RQ1.

### *4.1. PaTyBRED*

Our proposed approach is inspired by the Path Ranking Algorithm (PRA) [27] and SDValidate [52]. It consists of a binary classifier for every relation which predicts the existence of a given pair of subject and object in the given relation. The set of classifiers can be thought of as a single multilabel classifier with binary relevance (i.e., each relation that can hold between a pair of instances is a label), where one binary classifier is learned for each class separately, and local feature selection [39], with different classifiers being able to work on different sets of specialized features.

We use two kinds of features. The first one are the types of subject and objects. This kind of information has been successfully used for error detection in SDValidate [52]. By analyzing the types of subject and object in one given relation, one can easily spot a very common kind of error without relying on the domain and range restrictions, which are often inexistent or too general. For example, in DBpedia the triple `recordedIn(I'm_a_Loser, Abbey_Road)` is wrong. `I'm_a_Loser` is a song by The Beatles from the album `Abbey_Road` and the relation `recordedIn` has domain `MusicalWork` and range `PopulatedPlace`. A song being recorded in an Album is a clearly wrong fact. At the same time, if the object were `Abbey_Road_Studio` of the type `Recording_Studio`, which is not a subclass of `PopulatedPlace`, the fact would also be wrong according to a method relying solely on types. If there are many facts where songs are recorded in recording studios, statistical methods such as SDValidate would be able to identify that such a pattern is common, and therefore unlikely to be wrong, despite the violation of range restriction, while a song recorded in album is uncommon, therefore likely to be an error. Hence, statistical approaches such as SDValidate respect the actual usage of the ontology, rather than its axiomatic design. Recent works have been proposed that pinpoint such mismatches automatically [49].

The main problem with this kind of approach is that it solely relies on type features. That means such approaches do not work on knowledge graphs with no type assertions, and may have poor performance on datasets with a shallow type hierarchy, with non informative types, or with incomplete type assertions. Moreover, solely using type features, it is impossible to detect wrong facts with wrong entities of correct types, for instance, when a person instance is confused with another of same or similar name.

Alternatively, we can use path features similar to those of PRA. However, solely relying on path features also may lead to different problems. One of those issues is that correct facts may be labeled as errors because of incompleteness. For instance, if river instances have the properties `country` (i.e., the countries a river passes through, typically multi-valued), and `mouthCountry` (i.e., the country where the river's mouth is, typically single-valued), then the feature `country` will be relevant for the relation `mouthCountry` since the confidence of the rule $\texttt{mouthCountry}(X,Y) \Rightarrow \texttt{country}(X,Y)$ is close to 1. However, some rivers do not have any assertions for `country` because of incompleteness, thus their correct `mouthCountry` assertion is

predicted to be wrong. That can lead to propagation of incompleteness.

Another problem is that since `country` is a more relevant feature to `mouthCountry` than vice versa, since the latter is far less common than the former. Hence, if an error occurs in the assertion of `country` for a river, it might happen that a correct `mouthCountry` assertion ends up being more likely to be detected as an error than the wrong `country` assertion.

In order to make our approach more robust and rule out issues caused by the two approaches, we combine both type and path features.

Finding the relevant paths for each relation can be a challenging task. Since several paths may be relevant for different relations, we compute all possible paths up to a given length, and for every relation's local classifier, we perform local feature selection. The number of possible paths grows exponentially with the number of relations, therefore an exhaustive search can easily become unfeasible. It is then crucial to have heuristics to efficiently navigate the search space. In the following subsection we propose and discuss such heuristic measures.

### *4.2. Extracted Features*

Our method includes the following parameters that define the path selection: maximum path length, maximum number of paths per length, and path selection heuristics. Following the approach described in [27], we use the domain and range restrictions of relations for pruning uninteresting paths, and we do not allow a relation to be immediately followed by its inverse. If the number of possible paths of a certain length exceeds the maximum number of paths per length, we apply our path selection heuristics to prune the least interesting paths and comply with the specified paths upper limit.

We define a path $P$ as a sequence of relations $r_1 \rightarrow ... \rightarrow r_i \rightarrow ... \rightarrow r_n$. The sequence of relations is connected by a chain of variables, with $P(s,o)$ meaning $s$ and $o$ can be connected by a path $P(s,o) \iff r_1(s,x_1) \wedge ... \wedge r_i(x_{i-1},x_i) \wedge ... \wedge r_n(x_{n-1},o)$. The inverse of a relation $r$ is denoted as $r^{-1}$ where $r^{-1}(s,o) = r(o,s)$ can also be part of paths. A path of length one $P = (r)$ is equivalent to the relation itself, i.e., $P(s,o) \equiv r(s,o)$. The length of a path is denoted as $|P|$. We define the set of subjects of $P$ as $s_P = \{s|P(s,o)\}$ and set of objects as $o_P = \{o|P(s,o)\}$.

PaTyBRED supports type and path features. The features for learning the classifier for a relation $r$ are shown in Table 1, where each instance is a pair of a subject and an object entity $(s,o)$, $X$ is a variable which can be any entity, and $p = r_1 \rightarrow ... \rightarrow r_n$ is a path of length $n \in \{1,...,mpl\}$, where $mpl$ denotes the maximum path length.

| Feature | Description | Condition |
|---|---|---|
| $C(s)$ | type of subject | $\exists r.\top \sqsubseteq C$ |
| $C(o)$ | type of object | $\exists r^{-1}.\top \sqsubseteq C$ |
| $p(s,o)$ | relations path subsumption | $r \sqsubseteq p$ |
| $p(X,s)$ | ingoing path from subject | $\exists r.\top \sqsubseteq \exists p.\top$ |
| $p(s,X)$ | outgoing path from subject | $\exists r.\top \sqsubseteq \exists p^{-1}.\top$ |
| $p(X,o)$ | ingoing path from object | $\exists r^{-1}.\top \sqsubseteq \exists p.\top$ |
| $p(o,X)$ | outgoing path from object | $\exists r^{-1}.\top \sqsubseteq \exists p^{-1}.\top$ |

Table 1
Kinds of binary features supported by PaTyBRED

Relations and paths can be represented as adjacency matrices of size $|N_I| \times |N_I|$.The adjacency matrix of $P$ can be computed by the dot product of its relations. However, computing the dot product of adjacency matrices can be an expensive operation, especially in large-scale knowledge graphs with millions of entities and high number of relations. Therefore, we need heuristic measures to prune the search space and compute the dot product only for the most relevant paths.

Let $A$ and $B$ be adjacency matrices – which can refer to a single relation or a path – which we want to concatenate in order to form a new path $A \cdot B$. Hence, we require a heuristic measure which can estimate the relevance of the path $A \cdot B$ *without* having to perform a potentially expensive full matrix multiplication to compute its adjacency matrix. Since the paths computed are to be used by all relations, the proposed heuristic measures should not be computed with respect to a target relation, but only consider the matrices $A$ and $B$.

Paths with empty adjacency matrices ($|A \cdot B| = 0$) are useless and should be pruned. A simple way to safely prune them is to calculate $o_A \cap s_B$. The set of objects $o_A$ contains the columns of $A$ which have non-zero elements, and the set of subjects $s_B$ contains the rows of $B$ which have non-zero elements. If the intersection is empty, then we know that $|A \cdot B| = 0$. Note that $|s_B| \leqslant |B|$ and $|o_A| \leqslant |A|$, and the intersection is cheaper to compute than dot product, therefore the runtime for computing $o_A \cap s_B$ is shorter.

While paths with empty adjacency matrices can be pruned safely without information loss, paths with very sparse, yet non-empty adjacency matrices are less likely to be informative for the classifier. Hence, we

apply a less defensive pruning and define heuristics for pruning paths with sparse adjacency matrices. Since the size of the intersection $o_A \cap s_B$ can be a good indicator of the number of nonzero elements in $A \cdot B$, we use it to define three measures for estimating the relevance of a path $A \cdot B$: We employ that characteristic into three proposed relevance measures *inter*, $m1$ and $m2$ (c.f Equations 1, 2 and 3).

$$inter(A,B) = |o_A \cap s_B| \tag{1}$$

$$m_1(A,B) = \frac{|o_A \cap s_B|}{|s_A \cap o_B| + 1} \tag{2}$$

$$m_2(A,B) = |o_A \cap s_B| \times |s_A \cup o_B| \tag{3}$$

For each length, a only a fixed number of paths is kept, which is a parameter in our approach (*mppl*). Hence, only the best scoring paths are used for creating longer paths, as well as for creating features to be used by the classifier. By early pruning irrelevant paths, time is saved not only by computing fewer adjacency matrices, but also the number of features to be considered is reduced (fewer columns in the features table to be populated and less features to have the relevance computed).

Once the relevant paths have been selected, we compute their adjacency matrices and use them to populate the features used to train the relation classifiers. One of the problems of computing the whole adjacency matrix of paths is that some can be very dense and require a lot of memory. For example, the path `birthPlace` $\rightarrow$ `locatedIn`$^{-1}$ on DBpedia, which represents everything which is located in a place where someone was born. Its adjacency matrix contains around 100 million nonzero elements and consumes more than 1GB of memory. As it is unlikely that all the entries in the matrix will be used, it would be desirable to handle such cases in a more efficient manner in order to restrict the memory consumption and speed up the paths adjacency matrices computation process.

It is worth pointing out that the `rdf:type` relation is not considered in the paths. Types are treated separately and are used to generate the type features, which consist of the set of asserted and subsumed types of an instance (we materialize the subsumed types into the assertions and ignore the subsumption relations). Integrating types into the paths can be problematic. Firstly, it would significantly increase the search space. Secondly, a path which begins with the property `rdf:type` can only continue with `rdf:type`$^{-1}$, because types can only be objects in this relation (if we do not consider OWL class axioms in paths), and as mentioned earlier, we do not allow a relation to be immediately followed by its inverse.

### *4.3. Learning the Model*

Once the paths have been selected, and their adjacency matrices have been computed, we can use them together with types as features to predict the existence of an entity pair $(s,o)$ in a relation. The first step is to build a training dataset containing all extracted features for each relation $r$. We use as positive examples the entity pairs $D_{\text{pos}} = \{(s,o)|r(s,o)\}$, i.e. all the non-zero cells in the relation's adjacency matrix. Following [4], we generate negative instances $D_{\text{neg}} = \{\gamma(s,o)|(s,o) \in D_{\text{pos}} \wedge \gamma(s,o) \notin D_{\text{pos}}\}$ for supervised training by corrupting entity pairs using a function $\gamma$, which substitutes the subject or the object with a random entity instance, ensuring the new pair is not positive. In a preliminary experiment, we compared this approach with that of [27], which is more expensive, and no significant difference in performance was observed.

As label, we use information from $r$ indicating the existence of $(s,o)$ in the relation. We extract path features from $\mathcal{A}_R$ and type features from $\mathcal{A}_C$. The path features are boolean values indicating whether a path connects $s$ to $o$ ($P(s,o)|\forall P \in \mathcal{P} - (r)$). The type features consist of the types of $s$ and $o$ (including subsumed types), i.e. $\{C|C(s)\}$ and $\{C|C(o)\}$. Other possible path features include the existence of a path starting or ending in $s$ and $p$ ($P(s,X)$, $P(X,s)$, $P(o,X)$, $P(X,o)$) as proposed in SFE [18], however the authors found out that this kind of feature does not improve performance. We conducted a preliminary experiment, which confirmed their results, and therefore, we do not consider this kind of features in our approach.

In order to clarify how the relation classifiers are trained, Table 2 depicts provide a simple example of training data for the relation `livesIn`, containing six features. We assume the example data contains instances of the types `Person` and `Place`, and relations `livesIn`, `bornIn`, `child`, and `spouse` (which is symmetric). For the last two relations we have the following assertions: `child(Trump, Ivanka)`,

`child(William, George)`, `child(Kate, George)`, `spouse(Trump, Melania)` and `spouse(William,Kate)`. There are in total six assertions for the relation `livesIn`, therefore six positive examples in the training data. We generate one negative example for every positive ($nneg = 1$) by corrupting the subject or object by substituting either with a random entity.

Before we learn the local classifiers, we evaluate the relevance of the features. Since different features might be relevant for different relations, we perform feature selection separately for every relation. This allows the relation classifiers to work on a small set of locally relevant features, and, at the same time, removes irrelevant features which might act as noise and reduce the classifier's performance [39]. We use the filter method, which simply select the top-$k$ most relevant features, with $\chi^2$ as relevance measure.

Algorithm 1 shows how PaTyBRED works. The function RELEVANT_RELATIONS searches adds the inverse of non-symmetric relations and eliminates relations which do not satisfy the minimum support threshold. After the paths of each length $\ell$ up to the maximum path length (*mpl*) are selected. First the function PATH_RELEVANCE gets the top-*mppl* most relevant paths according to the selected path relevance measure. Once the paths are selected their adjacency matrices are computed and saved for later use.

Subsequently the relation classifiers need to be trained. For every relation $r$ the positive $(s, o)$ pairs are obtained with GET_POSITIVES, then a sample of size $fs_{size}$ is selected for feature selection and the negative examples are generated by corrupting the positives sample with GENERATE_NEGATIVES. Then a features table $X_{fs}$ and binary vector of labels $y_{fs}$ is generated with CREATE_FEATS_LABELS, with which the set of best features $feats[r]$ is selected. Finally a training features and labels are generated for a different set of sample of positives (of size $ts_{size}$) and the classification model $clf$ is trained with FIT_MODEL.

When comparing PaTyBRED with PRA and SFE, our approach has the following advantages:

- By decoupling the feature extraction and the learning step, we can use different popular classifiers to learn the relations, and we found indeed that logistic regression, which is used in PRA and SFE, is not the best performer.
- We introduce a local feature selection step prior to training the relation classifiers, which can significantly increase the computational performance.

**Algorithm 1** The PaTyBRED algorithm

1: **function** LEARN_PATYBRED_MODEL($R$, $A$, domains, ranges, *mppl*, *mpl*, $k$, *minsup*, *phsm*, $ts_{size}$, $fs_{size}$)
2: $R_{rel} \leftarrow$ RELEVANT_RELATIONS($R$, *minsup*)
3: $paths[0] \leftarrow R_{rel}$
4: $\ell \leftarrow 1$
5: **while** $\ell <$ *mpl* **do**
6: $rel \leftarrow \{\}$
7: **for** $p \in paths[\ell]$ **do**
8: **for** $r \in R_{rel}$ **do**
9: **if** $p[-1] \neq r^{-1} \wedge \text{range}[p[-1]] \cap \text{domain}[r] \neq \emptyset$ **then**
10: $p_{new} \leftarrow (p, r)$
11: $rel[p_{new}] \leftarrow$ PATH_RELEVANCE($A[p]$, $A[r]$, *phsm*)
12: **end if**
13: **end for**
14: **end for**
15: $paths_{best} \leftarrow$ SELECT_BEST_PATHS(*rel*,*mppl*)
16: **for** $p \in paths_{best}$ **do**
17: $A[p] \leftarrow A[p[:-1]] \cdot A[p[-1]]$
18: **end for**
19: $paths[\ell + 1] \leftarrow paths_{best}$
20: $\ell \leftarrow \ell + 1$
21: **end while**
22: $models \leftarrow \{\}$
23: $feats \leftarrow \{\}$
24: **for** $r \in R$ **do**
25: $so_{pos} \leftarrow$ GET_POSITIVES($r$, $A$)
26: $so\_fs_{pos} \leftarrow$ SAMPLE($so_{pos}$, $fs_{size}$)
27: $so\_fs_{neg} \leftarrow$ GENERATE_NEGATIVES($so\_fs_{pos}$, $A[r]$)
28: $X_{fs}$, $y_{fs} \leftarrow$ CREATE_FEATS_LABELS($so\_fs_{pos}$, $so\_fs_{neg}$, $paths \cup s_{types} \cup o_{types}$, $A$)
29: $feats[r] \leftarrow$ SELECT_FEATURES($X_{fs}$, $y_{fs}$, $k$)
30: $so\_train_{pos} \leftarrow$ SAMPLE($so_{pos}$, $ts_{size}$)
31: $so\_train_{neg} \leftarrow$ GENERATE_NEGATIVES($so\_train_{pos}$, $A[r]$)
32: $X$, $y \leftarrow$ CREATE_FEATS_LABELS($so\_train_{pos}$, $so\_train_{neg}$, $feats[r]$, $A$)
33: $models[r] \leftarrow$ FIT_MODEL($clf$, $X$, $y$)
34: **end for**
35: **return** *models*, $feats$
36: **end function**

- We propose heuristic measures to explore the paths search space, again for gaining computational peformance.

Moreover, negative evidence features, i.e. paths which connect negative but no positive entity pairs of a relation, are also considered. Since our approach is supervised and includes negative examples in the training data, this kind of features is extremely important to identify wrong facts.

## 5. Error Detection Experiments

In this section, we first briefly present the datasets used in the evaluations, then we present the experiments conducted, which are split into two parts. In the first part we perform an automatic evaluation to compare PaTyBRED with SDValidate and state-of-the-art link prediction methods, and in the second we conduct a manual evaluation of PaTyBRED on three large-scale datasets (DBpedia, NELL and YAGO) with actual er-

| | Features | | | | | | Label |
|---|---|---|---|---|---|---|---|
| $(s, o)$ | Person(s) | Place(s) | Person(o) | Place(o) | spouse → livesIn | child → bornIn | livesIn |
| (Trump,DC) | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| (Melania,DC) | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| (Ivanka,DC) | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| (William,London) | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| (Kate,London) | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| (George,London) | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| (NY,DC) | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| (Melania,Paris) | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| (Ivanka,Obama) | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| (Bill,London) | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| (Kate,Tokyo) | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| (Xi,London) | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

Table 2

Example of training data instances for the relation livesIn

roneous relation assertions . The experiments are designed to answer research question RQ1.

### *5.1. Datasets*

In our experiments, we use a variety of knowledge graphs, some of which are clean, and others noisy. In the first part of our experiments we automatically evaluate the performance of the error detection algorithms. In order to make the evaluation automatic, we use a variety of datasets to which we add synthesized wrong facts. We generate the erroneous facts by corrupting the subject or object f true facts, i.e., replacing the original entity with a randomly selected which results in a fact which does not exist in the original data. For our generation process, we corrupt 1% of the triples, using two different kinds of errors:

- For *type 1* errors, we corrupt the triple by substituting the object with any entity from the knowledge graph (independent of its type).
- For *type 2* errors, we corrupt the triple by substituting the object with any entity from the knowledge graph which has the same type(s).

That means the errors of the second kind are, in principle, more difficult to be detected than those of the first kind, since the new entity is more likely to have characteristics similar to those of the original one.[5]

The datasets used are the following: As input knowledge graphs, we use DBpedia (2015-10) [2], NELL (08m-690) [9], and YAGO3 [35]. We use the following smaller domain specific datasets: Semantic Bible[6], AIFB portal[7], and Nobel Prize[8]. Furthermore, we selected four of the largest conference datasets from the Semantic Web dog food corpus[9], i.e., LREC2008, WWW2012, ISWC2013, and ESWC2015. In addition, WN18 and FB15k (WordNet 1.8 and a subset of Freebase with 15 000 entities), which have been widely used on link prediction experiments, are also used.

The Semantic Web dog food datasets are known to be correct and *locally* complete, i.e. no errors or missing relations *between contained entities*, therefore, the generated errors can be used as gold standard. We could not find any evaluation the of quality of AIFB, Semantic Bible, or Nobel Prize. Since we cannot guarantee the quality of the data, the synthesized errors can be considered a silver standard.[10] The silver standard may contain both false positives (due to incompleteness of the underlying knowledge graphs), as well as false negatives (due to noise in the original knowledge graphs).

The number of false positives is likely to be low even for highly incomplete datasets, since in general, the number of missing facts is significantly smaller than the number of possible facts ($|N_R||N_I|^2 - |\mathcal{A}_R|$) from which the generated wrong facts are drawn.

[5] It should be noted that, although type 2 errors are, in theory, a subset of type 1 errors, the sets of errors added to the testsets are not subsets of each other. The probability of generating a type 1 error which is also a type 2 error depends on the distribution of types and differs from datasets to dataset; for the datasets used in our evaluation, it falls into a range between 0.13 and 0.32. However, a correlation of that probability with the approach's performance on the different dataets could not be observed.

[6] http://www.semanticbible.com/

[7] http://www.aifb.kit.edu/web/Web_Science_und_Wissensmanagement/Portal

[8] http://www.nobelprize.org/nobel_organizations/nobelmedia/nobelprize_org/developer/manual-linkeddata/terms.html

[9] http://data.semanticweb.org/dumps/conferences/

[10] We follow the notion that a *gold standard* is guaranteed to contain only correct examples (i.e., in our case, the labels for correct and incorrect triples are always accurate), whereas a *silver standard* may also contain a small fraction of incorrect examples (i.e., in our case, correct triples labeled as incorrect, or vice versa).

In the second part of the experiments, we use DBpedia and NELL as large-scale real-world use cases. These datasets are known to be noisy and incomplete, with type assertion completeness estimated to be at most 63.7% on DBpedia [52]. We do not synthesize any erroneous facts, and rank all the facts by their confidence values. Since we do not know the noisy facts or even the number of errors which exist in DBpedia, we manually evaluate the top-100 results.

In our experiments, we evaluate the impact of different parameter settings in our approach, and compare it with SDValidate and embedding-based knowledge graph completion methods. We use ProjE[11] as well as the TransE and HolE implementations of scikit-kge[12]. The three approaches are chosen as representatives of different flavors of embedding methods, each of which has been shown to yield good results on at least one knowledge graph completion task in the past and outperforming a number of other methods. [22, 26]. Those knowledge graph completion methods generally assign a score to a non-existing triple (i.e., a combination of a subject, predicate, and object *not* present in the knowledge graph), and for the completion task, the top scoring triples are considered as useful completions for the knowledge graph. In order to use those methods for *error detection*, we make use of the same scoring mechanism, but apply it to *existing* triples. Low-scoring triples are considered erroneous.

Furthermore, to analyze the benefits of combining path and type features, we also compare against the variants of PaTyBRED using only path features (*PaBRED*) and only type features (*TyBRED*). For that reason, we omit a direct comparison our method with SFE, since, by design, PaBRED performs at least as good as SFE. The implementation of PaTyBRED, as well as the SHACL constraint generation is available on Github[13].

The reported results from the embedding methods were obtained by not considering the type assertions. We tried adding the type assertions as an extra relation, however, this did not improve the results. The embedding methods suffer from the problem that the distribution of scores over different relations is not uniform. Often some relations have average triple scores lower than others, and this can result in a bias when detecting errors.

[11] https://github.com/nddsg/ProjE
[12] https://github.com/mnick/scikit-kge
[13] https://github.com/aolimelo/kged

In order to address this problem, we use the following strategy to normalize the scores across different relations: in a first step, we run the isolation forest outlier scoring algorithm [34] to detect outliers in the confidence values of each relation separately. We then use the outlier scores instead of the triple confidence values to rank the facts, since they share a common global scale. Since unusually high confidence values are also outliers and we are interested only in the outliers of low scores, we do not consider as outlier any fact with score greater than the relation's average.

### 5.2. Evaluation Metrics

For the error detection problem, we use ranking measures to evaluate the performance of the error detection algorithms, since we compute scores for every triple in the graph and generate a ranking. More specifically, we generate an error score for each triple, and we rank the triples by that error score. With that ranking, ideally all erroneous triples should be ranked higher than the correct ones. We use the mean rank ($\mu$R) and mean reciprocal rank (MRR):

$$\mu R = \frac{1}{|E|}\sum_{i=1}^{|E|} rank_i \tag{4}$$

$$MRR = \frac{1}{|E|}\sum_{i=1}^{|E|} \frac{1}{rank_i} \tag{5}$$

One shortcoming of those metrics is that they are not comparable across datasets.

To illustrate those shortcomings, table 3 shows a toy example with the rankings of two approaches on two datasets. While approach 1 is perfect and ranks all errors (E) higher than all correct relations (C), approach 2 makes some mistakes. As we can observe in this example, the $\mu$R and MRR are not comparable across datasets of different sizes: approach 2 has a the same $\mu$R and a better MRR on dataset 1 than approach 1 on dataset 2, although the results are actually worse.

To overcome those shortcomings and make the results comparable, we use the filtered variants f$\mu$R and fMRR (c.f. Equations 6 and 7), which filter out correctly higher ranked predictions:

$$fMRR = \frac{1}{|E|}\sum_{i=1}^{|E|} \frac{1}{rank_i - i + 1} \tag{6}$$

| | Dataset 1 (5 instances) | | Dataset 2 (10 instances) | |
|---|---|---|---|---|
| | Appr. 1 | Appr. 2 | Appr. 1 | Appr. 2 |
| | E | E | E | E |
| | E | C | E | C |
| | E | E | E | C |
| | C | C | E | E |
| | C | E | E | C |
| | – | – | C | E |
| | – | – | C | E |
| | – | – | C | C |
| | – | – | C | E |
| | – | – | C | C |
| $\mu$R | 2 | 3 | 3 | 5.4 |
| MRR | 0.61 | 0.51 | 0.45 | 0.33 |
| f$\mu$R | 1 | 2 | 1 | 3.4 |
| fMRR | 1 | 0.61, | 1 | 0.4 |

Table 3
Toy example showing rankings of two error detection approaches on two datasets

$$f\mu R = \frac{1}{|E|}\sum_{i=1}^{|E|} rank_i - i + 1 \tag{7}$$

Subtracting $i-1$ from the rank ensures that better ranked true positives are filtered out. As we can observe in the example, the best approaches always score 1 for the f$\mu$R and the fMRR, with the inferior approaches being consistently ranked worse. Hence, those filterings can be used to make results comparable across datasets of different sizes and with different error rates.

### *5.3. Parameter Settings*

First, we evaluate how the different PaTyBRED parameters affect its performance. The evaluated parameters are the maximum path length (*mpl*), the maximum number of paths per length (*mppl*), the path selection heuristic measure (*pshm*), the number of locally selected features ($k$), and the local classifier ($clf$).

As far as the maximum path length (*mpl*) is concerned, the best results were achieved with $mpl = 2$, that is direct links and triangular patterns. Equivalent, inverse, and subproperty relations, as well as other kinds of associations can be exploited with direct links, while more complex associations with composed relations can be exploited with the triangular patterns. Examples of direct link and triangular pattern for the relation `livesIn` are respectively `bornIn` and `playedFor/locatedIn`.

In none of the datasets used in our experiments, a $mpl > 2$ achieved better results. It seems that paths longer than two do not bring any information gain, while it significantly increase the search space and slows runtime.

In our experiments, we evaluate three different classifiers ($clf$): random forests (RF) [6], support vector machines (SVM) [12] and logistic regression (LR). We also try two different number of selected features $k$, i.e., $k = 10$ and $k = 25$. These numbers are low because we observed that only a small number of path and type features are relevant to the local relation classifiers. Table 4 shows how the different settings of PaTyBRED $^{clf}_{k}$ on various datasets. The results show that RF and SVM achieved the best results, while LR – which is used in PRA and SFE – lagged behind.

The heuristic measures used for selecting relevant adjacency matrices are those proposed in Section 4.2, i.e., *inter*, $m1$ and $m2$. As a baseline, we use the *random* selection of paths. In order to better evaluate the quality of the paths selected we exclude the type features and consider exclusively the selected paths. We compared the heuristic measures on all the datasets presented in Section 5.1, ranked the measures and averaged them, as advised in [14]. In order to find out the significance of the results we perform Nemenyi Test with $\alpha = 0.05$. Since the number of datasets is rather small, the difference between *inter* and $m2$ is not significant, however, they are significantly better than the random approach (c.f. Figure 3).[14] Given those results, we set the maximum paths per length (*mppl*) to 1,000 and use $m2$ as a heuristic measure when the maximum number of paths exceeds 1,000.

### *5.4. Comparison*

Tables 5 and 6 report a comparison between PaTyBRED and the other state-of-the-art models. Table 5 refers to errors generated by replacing entities with entities of arbitrary types (*errors of kind 1*). Table 6 refers

[14]The diagram is to be read as follows: the x axis depicts the average rank of the different approaches across different datasets. A higher ranked approach which is more than the critical distance (CD) away from a lower ranked one outperforms the lower ranked one *statistically significantly*. The black bars groups approaches whose performance differences are *not* statistically significant. This means that in this diagram: m2 and inter significantly outperform random, while m1 does not. The difference between m1, m2, and inter is not significant.

| | *fMRR* | | | | | | | | | *fμR* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | sembib | eswc | iswc | www | lrec | nobel | aifb | wn18 | fb15k | sembib | eswc | iswc | www | lrec | nobel | aifb | wn18 | fb15k |
| PaTyBRED $^{\text{LR}}_{10}$ | 0.800 | 0.835 | 0.811 | 0.212 | 0.754 | 0.690 | 0.014 | 0.584 | 0.618 | 0.008 | 0.020 | 0.006 | 0.0023 | 0.011 | 0.076 | 0.041 | 0.00352 | **0.015** |
| PaTyBRED $^{\text{RF}}_{10}$ | 0.840 | 0.927 | 0.933 | 0.559 | 0.747 | 0.680 | 0.120 | **0.860** | **0.770** | 0.009 | **0.009** | 0.010 | 0.0003 | 0.006 | 0.080 | 0.031 | **0.00003** | 0.018 |
| PaTyBRED $^{\text{SVM}}_{10}$ | 0.838 | 0.906 | **0.980** | 0.414 | **0.844** | 0.673 | 0.070 | 0.820 | 0.713 | 0.011 | 0.012 | 0.008 | 0.0007 | **0.004** | 0.103 | 0.041 | **0.00003** | **0.014** |
| PaTyBRED $^{\text{LR}}_{25}$ | 0.745 | 0.907 | 0.862 | 0.707 | 0.786 | **0.788** | 0.068 | 0.584 | 0.524 | 0.005 | 0.022 | **0.003** | 0.0012 | 0.011 | **0.051** | 0.035 | 0.00349 | **0.014** |
| PaTyBRED $^{\text{RF}}_{25}$ | **0.881** | **0.928** | 0.964 | **0.795** | 0.653 | 0.782 | **0.213** | 0.795 | 0.545 | **0.003** | 0.028 | 0.010 | **0.0001** | 0.006 | **0.051** | **0.028** | 0.00004 | 0.020 |
| PaTyBRED $^{\text{SVM}}_{25}$ | 0.848 | 0.860 | **0.980** | 0.537 | 0.822 | **0.788** | 0.045 | 0.570 | 0.765 | 0.007 | 0.015 | 0.006 | 0.0003 | 0.005 | 0.063 | **0.028** | 0.00006 | 0.014 |

Table 4

Comparison of local classifiers and number of selected features on generated errors of kind 1

| | *fMRR* | | | | | | | | | *fμR* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | sembib | eswc | iswc | www | lrec | nobel | aifb | wn18 | fb15k | sembib | eswc | iswc | www | lrec | nobel | aifb | wn18 | fb15k |
| PaTyBRED | **0.881** | **0.928** | **0.980** | **0.795** | **0.844** | **0.788** | **0.213** | **0.860** | **0.770** | **0.003** | **0.009** | **0.003** | **0.0001** | **0.004** | **0.051** | **0.028** | **0.00003** | **0.014** |
| TyBRED | 0.463 | 0.782 | 0.315 | 0.744 | 0.693 | 0.758 | 0.205 | — | — | 0.121 | 0.083 | 0.102 | 0.0740 | 0.113 | 0.084 | 0.085 | — | — |
| PaBRED | 0.800 | 0.831 | **0.980** | 0.503 | 0.778 | 0.200 | 0.173 | **0.860** | **0.770** | 0.009 | 0.010 | 0.005 | 0.0008 | **0.004** | 0.227 | 0.056 | **0.00003** | **0.014** |
| SDValidate | 0.265 | 0.140 | 0.218 | 0.109 | 0.307 | 0.464 | 0.022 | — | — | 0.355 | 0.397 | 0.326 | 0.3768 | 0.339 | 0.286 | 0.293 | — | — |
| ProjE | 0.102 | 0.175 | 0.047 | 0.098 | 0.138 | 0.187 | 0.048 | 0.004 | 0.014 | 0.149 | 0.197 | 0.201 | 0.1796 | 0.179 | 0.177 | 0.252 | 0.18714 | 0.125 |
| HolE | 0.011 | 0.018 | 0.025 | 0.018 | 0.065 | 0.026 | 0.001 | 0.002 | 0.006 | 0.204 | 0.258 | 0.108 | 0.1170 | 0.108 | 0.213 | 0.235 | 0.17304 | 0.083 |
| TransE | 0.058 | 0.001 | 0.000 | 0.001 | 0.039 | 0.051 | 0.005 | 0.001 | 0.000 | 0.226 | 0.302 | 0.280 | 0.2381 | 0.163 | 0.320 | 0.329 | 0.26174 | 0.190 |

Table 5

Comparison of FMRR on generated errors of kind 1

| | *fMRR* | | | | | | | *fμR* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | sembib | eswc | iswc | www | lrec | nobel | aifb | sembib | eswc | iswc | www | lrec | nobel | aifb |
| PaTyBRED | 0.482 | 0.553 | **0.941** | 0.609 | **0.532** | 0.022 | **0.272** | **0.082** | 0.124 | 0.023 | 0.035 | 0.027 | 0.250 | 0.080 |
| TyBRED | 0.001 | 0.001 | 0.001 | 0.001 | 0.000 | 0.000 | 0.000 | 0.597 | 0.503 | 0.512 | 0.495 | 0.551 | 0.526 | 0.496 |
| PaBRED | **0.579** | **0.567** | **0.941** | **0.625** | 0.486 | **0.250** | 0.205 | 0.086 | **0.099** | **0.017** | **0.023** | **0.011** | **0.212** | **0.065** |
| SDValidate | 0.001 | 0.001 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.570 | 0.457 | 0.467 | 0.506 | 0.495 | 0.495 | 0.475 |
| ProjE | 0.064 | 0.026 | 0.015 | 0.026 | 0.007 | 0.067 | 0.018 | 0.215 | 0.362 | 0.223 | 0.245 | 0.254 | 0.274 | 0.269 |
| HolE | 0.022 | 0.015 | 0.043 | 0.049 | 0.059 | 0.053 | 0.004 | 0.240 | 0.324 | 0.192 | 0.190 | 0.192 | 0.294 | 0.246 |
| TransE | 0.092 | 0.004 | 0.012 | 0.000 | 0.012 | 0.001 | 0.003 | 0.247 | 0.308 | 0.239 | 0.337 | 0.148 | 0.413 | 0.339 |

Table 6

Comparison of FMRR on generated errors of kind 2

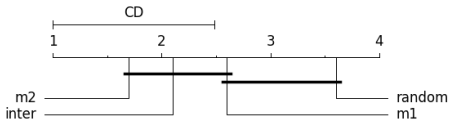



Fig. 3. Critical distance diagram comparing path selection heuristics

to errors where entities have been replaced by entities with the same types as the original entity (*errors of kind 2*). Table 6 does not contain results for WN18 and FB15k because the original datasets do not contain entity types, which prevents errors of kind 2 to be generated. For the same reason the results of SDValidate and TyBRED in Table 5 are not reported for WN18 and FB15k. We report values for $fMRR$ and $f\mu R$. To make the results on knowledge graphs of different sizes more comparable, the $f\mu R$ are values divided by the total number of facts in the KG.

It is noticeable that the results for AIFB are significantly worse than other datasets. One of the reasons is the fact that it has no inverse relations, which can be extremely helpful on the error detection. Another reason is the fact that in AIFB the `author` is defined by 27 `author_n` relations, with $n$ indicating the position in the authors list. That means it is necessary to not only model the author relation, but also all the $n^{\text{th}}$-author relations.

We can observe some larger variations in and between the datasets. The smaller sets, like *nobel* or *aifb*, do not have enough training information for some ap-

proaches, which work better on the larger *wn18* and *fb15k* datasets. The same holds for SDValidate, which is relying on larger datasets to create stable statistical distributions – in fact, SDValidate even has a hard coded switch that prevents it from reporting errors based on small distributions and little evidence to avoid false negatives. On the other hand, the classifiers used in PaTyBRED and its variants can learn stable models also for smaller datasets. Moreover, the embedding based approaches TransE, HolE, and ProjE, which have been developed for link *prediction* on large datasets, tend to overfit when it comes to link *validation*, especially for smaller scale datasets. Although there are quite a few successors and alternatives to the embedding based approaches tested here, the difference is so large that we do not expect a larger shift when trying more different embedding based methods.

As discussed above, PaTyBRED, TyBRED and PaBRED were run with 6 different configuration: $clf \in \{\text{LR,RF,SVM}\}$ and $k \in \{10, 25\}$. For each dataset, the results of the best performing configuration are reported. The values reported for the embeddings methods were the best for each dataset amongst number dimensions $d \in \{5, 15, 50, 100, 200\}$ and with the outlier detection, as explained earlier.

It is worth mentioning that the score normalization via outlier detection helped improve the performance of embeddings' $f\mu R$ performance on average on 15%. The best results for the embedding methods were obtained with $d = 15$ or $d = 50$ depending on the dataset. The results reported for the knowledge graph completion in the original paper for ProjE on FB15k were with $d = 200$. On error detection with the same dataset the best performance was with $d = 50$, cutting the $f\mu R$ in half. Additionally, $d = 5$ and $d = 15$ also had better performance than $d = 200$. This indicates that when using embeddings for error detection, the dimensionality should be lower than for KGC. Since the dataset contains wrong triples, which shouldn't be fit by the model, overfitting can severely affect the performance (more than underfitting).

Our proposed method outperforms all the other methods, with the embedding methods having a surprisingly low performance. PaTyBRED performs best when combining types and paths, with TyBRED (with types only) and PaBRED (with paths only) being generally worse. To further understand the importance of combining path type features, we analyze what kind of features are selected on the local classifiers and report the proportion of types and paths. Table 7 shows the average proportion of selected features over all relation classifiers with $k = 10$. Overall more type features are selected, but both kinds of features are relevant on the evaluated datasets. WN18 and FB15k are absent because they do not have type assertions, and therefore have only path features.

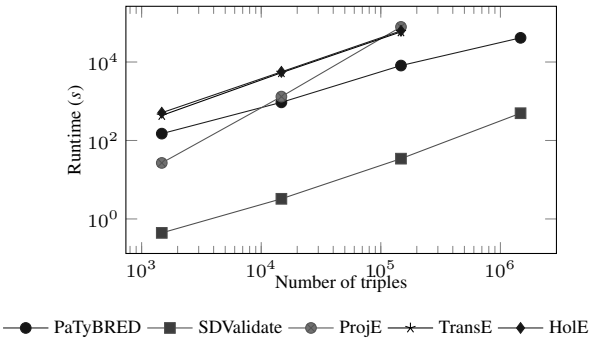



Fig. 4. Runtime comparison of the evaluated methods

Table 6, where the erroneous facts contain wrong instances of correct types, shows how the performance of methods which rely on types exclusively (SDValidate and TyBRED) is similar to that of random ranking with $f\mu R$ around 0.5. It also shows how detecting errors of kind 2 is more difficult than those of kind 1, and it reveals the importance of using path features for detecting facts with wrong instances of correct types. We can also observe that PaBRED has performance similar to PaTyBRED and even better on some datasets for kind 2 errors, since type features are useless to detect those errors, and not considering type features ensures that these cannot potentially replace more useful path features. The only exceptions are on LREC and AIFBportal, where PaTyBRED has better $fMRR$ than PaBRED. However, on the same datasets PaBRED performs better in terms of $f\mu R$, meaning that it has better average rank but less highly ranked instances.

In addition to evaluating the result quality, we also conducted a scalability study of the evaluated methods. The scalability test is performed on synthesized replica of DBpedia with the M3 model [40] of sizes {0.01%, 0.1%, 1% and 10%} of the original size, that means the number of triples varies from around 1.5k to 1.5M triples. The results are shown in Figure 4.

We can observe that SDValidate has by far the lowest runtimes, since it is a simpler model than the others. Amongst the embedding methods, ProjE which directly optimizes the rankings in the link prediction task, has the steepest runtime growth. HolE and TransE have similar scalability being more scalable than ProjE. PaTyBRED, due to the aggressive local

| | sembib | eswc | iswc | www | lrec | nobel | aifb | nell | dbpedia | yago |
|---|---|---|---|---|---|---|---|---|---|---|
| Paths | 0.432 | 0.412 | 0.415 | 0.358 | 0.479 | 0.222 | 0.182 | 0.032 | 0.060 | 0.142 |
| Types | 0.568 | 0.588 | 0.585 | 0.642 | 0.521 | 0.778 | 0.818 | 0.968 | 0.940 | 0.858 |

Table 7
Proportion of path and type features selected

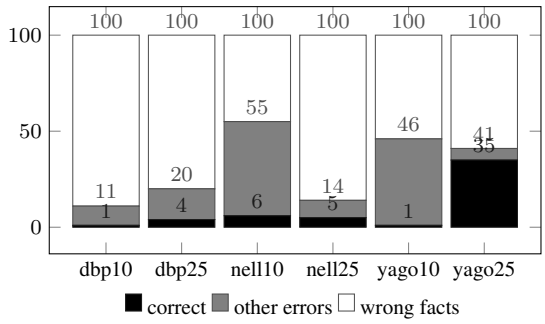



Fig. 5. Manual evaluation on DBpedia, NELL, and YAGO

feature selection and sampling, has the least steep of the curves, and, together with SDValidate, was the only approach to handle the larger knowledge graphs in less than 24 hours. This indicates the appropriateness of PaTyBRED for handling large datasets.

### 5.5. Manual Validation

In this section, we perform a manual validation of PaTyBRED on three large-scale knowledge graphs: DBpedia, NELL, and YAGO. We have a deeper look at the top-100 results and classify the triples as *correct*, *wrong* and *other errors*, i.e., correct triples with related errors, e.g., wrong or missing types of subject or object. Note that by analyzing the top-100 results, we do explicitly *not* draw a representative, random sample of 100 triples to validate the accuracy of our approach. We rather measure precision@100 of the approaches. This is similar to how a knowledge graph engineer would utilize the approach: they would typically not inspect random errors, but the ones in which the automated approach has the highest confidence.

The results are shown in Figure 5 with PaTyBRED $^{\text{RF}}_{10}$ and PaTyBRED $^{\text{RF}}_{25}$ on DBpedia (dbp10, dbp25), NELL (nell10, nell25) and YAGO (yago10, yago25). PaTyBRED seems to perform better on DBpedia and YAGO with less local features (10), and with more on NELL (25). Most of the other error cases occurred because of type assertion incompleteness, with the subject or object often having no types at all. Deleting these triples would lead to propagation of incompleteness. These cases could be automatically detected (i.e., by checking whether types are present for the subject and object), and some of them fixed if the type completion methods [41, 51] are combined with error detection. The quality of predicted types can be asserted by the improvement of the scores of triples containing the entities with predicted types.

Some of the errors come from mistakes when linking Wikipedia pages with very similar names. Such problems could potentially be evaluated with CoCKG. Section 6 presents the approach in more details and evaluates its performance on DBpedia and NELL.

Entities in DBpedia are described in much more detail than in NELL [57]. Around 20% of NELL's instances are untyped, while in DBpedia, only 1% of them have no types other than `owl:Thing`. Furthermore, in NELL, reasoning is already used in the construction process for error detection, which means that very obvious errors and violations of the underlying ontology are already removed. This may explain why NELL performs better with more locally selected features, as opposed to DBpedia. By increasing the number of features the number of correct facts with untyped subject or object in the top-100 was reduced from 48 to 9, and the number of actual errors increased from 45 to 86.

Amongst the five correct facts from DBpedia which were wrongly predicted to be errors, two were from the relation `seeAlso`. That is understandable since the relation has very wide semantics, and any pair of vaguely related entities can be correct, therefore, learning a model for such a relation may be very difficult. Another error detected was `location(Alan_Turing_Institute, British_Library)`, which is a correct fact, but the unique case of an organization which is located in a library. The last case is with the `foundedBy` relation, with two cases of newspapers found by political parties, not persons.

For YAGO, the results are considerably worse than for DBpedia and NELL. There are various reasons here: first, the schema of YAGO is very different, with only 77 relations, but 488,469 classes [35]. Hence, compared to DBpedia with 1,105 relations and 760 classes, the search space for path and type features is completely different – we cannot construct too many interesting paths, and many of the types are too spe-

cific to be meaningful for error detection. Second, the global error rate of YAGO is lower [15], with more sophisticated checking in place already during YAGO's construction process, which makes the error detection task inherently more difficult.

## 6. Correction of Errors Approach

Once erroneous relation assertions have been identified at a high level of confidence, they may be removed from the knowledge graph. In case a suitable replacement for the relation can be found, they may be also be *corrected* instead of removed. In this section, we discuss the CoCKG (Correction of Confusions in Knowledge Graphs) approach for finding suitable replacements for an erroneous relation assertion. The approach is designed to address research question RQ3.

The approach consists of first running an error detection algorithm (PaTyBRED in the case of this paper), selecting the top-$k$ facts most likely to be wrong. In the next step, the error is heuristically verified to be an actual relation assertion error and not caused by missing or wrong type assertions in the object or subject with a type predictor $tp$. In the final step, candidate entities are retrieved, and if any of the candidates significantly improves the likelihood of the triple being right, we replace it by that candidate. This idea is similar to using a relation prediction algorithm for scoring the candidates at hand. In both cases, the likelihood of a triple being correct is estimated and used to decide whether or not to perform the substitution.

The function CORRECT_TRIPLE in Algorithm 2 gives an overview of how CoCKG works. The parameter $T$ is the set of all triples in the knowledge graph, $T_{err}$ is the set of triple and confidence pairs generated by the error detection model (*ed*), $tp$ is the type predictor, *mc* is the minimum confidence threshold, and *mcg* the minimum confidence gain threshold, i.e. the ratio of the new and old triple scores. In the next subsections we discuss the other parts in more details.

**Algorithm 2** Knowledge base correction process

```
1:  function CORRECT_TRIPLES(T, T_err, ed, tp, mc, mcg)
2:      T_corr ← ∅
3:      for t, score_t ∈ T_err do
4:          s, p, o ← t
5:          s_tp ← PREDICT_TYPES(tp, s)
6:          o_tp ← PREDICT_TYPES(tp, o)
7:          if ¬(CONF_NT(ed, t, s, s_tp) ∨ CONF_NT(ed, t, o, o_tp)) then
8:              s_cand ← GET_CANDIDATES(s)
9:              o_cand ← GET_CANDIDATES(o)
10:             T_cand ← {(s_i, p, o)|s_i ∈ s_cand} ∪ {(s, p, o_i)|o_i ∈ o_cand}
11:             T_cand ← T_cand − T
12:             c_best, max_conf ← nil, conf
13:             for c ∈ T_cand do
14:                 if s ∈ domain(p) ∧ o ∈ range(p) then
15:                     score_c ← CONF(ed, c)
16:                     if score_c ⩾ mc ∧ score_c/score_t ⩾ mcg then
17:                         c_best, max_conf ← c, score_c
18:                     end if
19:                 end if
20:             end for
21:             if c_best ≠ nil then
22:                 T_corr ← T_corr ∪ {(c_best, t)}
23:             end if
24:         end if
25:     end for
26:     return T_corr
27: end function
```

### 6.1. Type Prediction

After selecting the $k$ triples most likely to be wrong, we first check if their confidence is low because of missing or wrong instance types (subject or object). In order to do that, we run a type predictor $tp$ on the subject and object instances. In this paper, we use as $tp$ a multilabel random forest classifier based on qualified links (i.e. ingoing links paired with subject type and outgoing links paired with object type), as described in [41]. If the set of predicted types of the subject are different from the actual types, we change the type features used by *ed* and compute a new confidence for the triple (c.f. CONF_NT). If the new score satisfies *mc* and *mcg*, then we conclude that the error was in the subject type assertions. The same is done for the object.

If in neither case (i.e., after recomputing the confidence with changed types for the subject and the object) the confidence thresholds are satisfied, we assume that the triple is actually wrong (i.e., a true negative), and not identified as erroneous by mistake (i.e., a false negative). In that case, we proceed to the next part where we try to substitute the subject and object with their respective lists of candidates.

Combining the type prediction process with the error detection also has the advantage that the newly predicted types can be validated on triples containing the instance whose types were predicted. This can help support, or contradict the type predictor, possibly detecting types which are wrongly predicted by identifying triples where the score is lowered with the new types.

### 6.2. Retrieving Candidates

As discussed above, we assume that one common source of erroneous assertions is the confusion of entities with similar names. Hence, a simple way to find candidate entities to resolve entity confusions

is to use disambiguation pages in Wikipedia. However, since disambiguation pages are only available for Wikipedia-based knowledge graphs, and furthermore are not available for each entity (e.g. `Ronaldo` has no disambiguation page), and in some cases the disambiguation pages miss important entities (e.g. the page `Bluebird_(disambiguation)` misses the entity `Bluebird_(horse)`, hence, we cannot correct the fact `grandisre(Miss_Potential,Bluebird)`), we require an additional source of candidates.

Since in our experiments we consider DBpedia and NELL, which have informative IRIs (in the case of DBpedia extracted from the correspondent Wikipedia's page), we search for candidate entities which have similar IRIs. Alternatively, for knowledge graphs with non-informative IRIs (e.g., Wikidata or Freebase), we could pursue the same approach and search for entities with similar labels. In this paper, we refer to the informative part of an IRI as the "name" of the entity, and note that there might be other sources of a name, such as an entity label.

Retrieving all the instances of similar names can be a time consuming task. This kind of problem is known as approximate string matching, and it has been widely researched [45, 73]. For our method, we use an approximate string matching approach based on [43]. First, we remove the IRI's prefix and work with the suffix as the entity's name. We then tokenize the names and construct a deletions dictionary with all tokens being added with all possible deletions up to a maximum edit distance $d_{max}$ threshold. This dictionary contains strings as keys and lists with all tokens which can turn into the key string with up to $d_{max}$ deletions as values. Only pairs of tokens which share a common deletion string can have an edit distance less or equal than $d_{max}$. We also have a tokens dictionary which has tokens as keys and lists of entities which contain a given token as values. With that, given a token and a $d_{max}$, we can efficiently obtain all the entities which contain that a string approximately similar to that token up to the maximum edit distance.

When searching for entities similar to a given entity, we perform queries for every token of the entity's name and we require that all tokens are matched. That is, for a certain entity to be considered similar, it has to contain tokens similar to all the tokens of the queried entity. A retrieved entity may have more tokens than the queried entity, but not less. The idea is that in general, when referring to an entity, it is common to underspecify the entity, but highly unlikely to overspecify it. E.g., it is more likely that `Ronaldo` is wrongly used instead of `Cristiano_Ronaldo` than the other way around. Furthermore, it reduces the number of matched entities.

We also perform especial treatment on DBpedia and NELL entity names because of peculiarities in their IRI structures. In DBpedia it is common to have between parentheses information to help disambiguate entities, which we consider unnecessary since the entity types are used in the error detection method. In NELL the first token is always the type of the entity, therefore, for similar reasons, we ignore it.

### 6.3. Correcting Wrong facts

At this point, for each assertion identified as erroneous, we have a list of candidate entities for replacing the subject and the objects, gathered, e.g., by exploiting disambiguation pages and approximate string matching. We then compute a custom similarity measure $s(e_1, e_2)$ between an entity $e_1$ and a candidate $e_2$. Each entity $e_i$ consists of a set of its tokens. The measure we propose consists of two components. The first is the sum of Levenshtein ($d_L$) distance of all matched tokens, and the second considers the number of unmatched tokens to capture a difference in specificity. The set of approximately matched token pairs is represented by $\mu(e_1, e_2)$ and the constant $c$ is the weight of the second component. This measure is used to sort the retrieved candidates, to prune them in case there are too many, and to break ties when deciding which of the top-scoring candidates should be chosen.

$$s(e_1, e_2) = \sum_{(t_1,t_2)\in\mu(e_1,e_2)} d_L(t_1, t_2) + c\frac{|e_1| - |\mu(e_1, e_2)|}{|e1|} \tag{8}$$

In case the relation has domain or range restrictions, we remove the candidates which violate these restrictions. Later, for each of the candidates, we generate triples by substituting the subject and object by each of the instances in its candidates lists (first substitute subject only, then object only). That is, the total number of candidate triples is the sum of the size of the subject and object candidates list. We do not create candidate triples by substituting both the subject and object at the same time because, although possible, we assume the simultaneous confusion of both instances

to be highly unlikely.[15] This restriction also limits the number of possible candidate triples to a linear instead of a quadratic number.

From the set of candidate triples, we remove those triples which are already existent in the KG. We compute the confidence of the remaining candidate triples and select that with highest confidence, given that *mc* and *mcg* are satisfied. As a result, CoCKG outputs a set of erroneous triples with a suggested replacement.

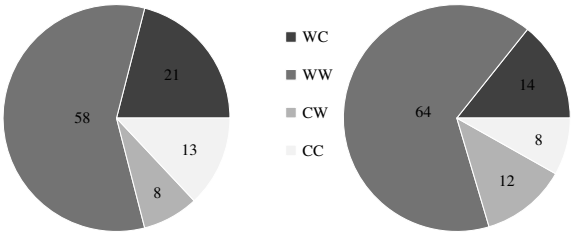



Fig. 6. Manual evaluation on DBpedia and NELL respectively

## 7. Correction of Errors Experiments

To validate the performance of error correction, and to answer research question RQ3, we conduct a manual evaluation on DBpedia (2016-10) and NELL (08m-690). For each knowledge graph, we have run PaTyBRED, and presented the top-1% most likely to be errors to CoCKG. We inspected the resulting corrections and classified them into four different categories:

1. WC: wrong fact turned into correct
2. WW: wrong fact turned into another wrong fact
3. CW: correct fact turned into wrong fact
4. CC: correct fact turned into another correct fact

Note that while *WC* is the only class that actually *improves* the knowledge graph, it does not mean that the other classes actually make it worse. In fact, only *CW* reduces the quality of the underlying knowledge graph, while *WW* and *CC* do not alter the amount of correct and wrong axioms in the knowledge graph.

Our approach was run with $mc = 0.75, mcg = 2$ and entity similarity measure with $c = 1.5$.[16] That resulted in 24,973 corrections on DBpedia and 616 correction on NELL. It also detected that 873 (569) errors were caused by wrong types in DBpedia (NELL). The relation of suggestions corrections between DBpedia and NELL, although the numbers are very different, reflects the relation of the overall number of axioms in both knowledge graphs [57], the relation of wrong types is not. One possible reason is that while types in DBpedia are often incomplete, they are rarely incorrect [51].

For the evaluation, since manually evaluating all these corrections would be impossible, we randomly select 100 suggestied corrections on each knowledge graph to perform the evaluation.

The results of our manual evaluation are shown in Figure 6. The proportion of facts successfully corrected (WC) was rather low. However, the majority of suggested replacements is WW (which does not alter the quality the of the knowledge graph), and only a small fraction (8% and 12%, respectively) are of the problematic category CW. These results show that the approach is at least capable of making meaningful suggestions, and can be used by experts to maintain the quality of a knowledge graph, although maybe not in a fully automatic setting.

When evaluating some relations individually, we notice that some of them achieve good results. E.g., the relations `sire`, `damsire`, `grandsire` and `subsequentWork` reaching more than 90% of successful corrections (case 1). The approach works well for these relations because horses are often named after other entities, and artists often have albums named after themselves, which makes confusions likely, but also fairly easy to detect.

One of the problems of our approach is that since it relies on PaTyBRED, which cannot find many relevant path features on DBpedia and NELL [38], it is difficult to distinguish between candidate entities of same type. For example, in NELL, the entity `person_paul` as object of `book_writer` relation is always corrected with `writer_paul_feval`.

The decision to generate candidate triples by corrupting either the subject or object seemed to have worked well for DBpedia, where we could not find a triple where both subject and object were wrong. On the other hand, in NELL such case was observed a few times, e.g. `ismultipleof`(`musicinstrument_herd`, `musicinstrument_buffalo`) whose object was cor-

[15] For that to happen in the case of DBpedia, a Wikipedia user would have to go to the wrong article page and insert a wrong link in the infobox. In NELL, an extraction would have to extract a relation by misinterpreting both involved entities at the same time, which, since reasoning, among other plausibliity checks, is involved in the creation of NELL, would require a plausible triple with a subject and object with a similar name *and* a compatible type, e.g., two football players and two football clubs with a similar name.

[16] The parameter values were selected based on heuristics and may not be optimal

rected to mammal_buffalo but the subject remained wrong.

Also, our assumption that confusions tend to use a more general IRI instead of a more specific, requiring all tokens of the queried to be matched, does not always hold. One example in DBpedia which contradicts this assumption is language(Paadatha_Thenikkal , Tamil_cinema), whose corrected object would be Tamil_language and could not be retrieved by our approach. While this can be a problem, dropping this assumption also means that more candidates entities will be retrieved, increasing the number of unrelated candidates, resulting in more candidate triples which need to be tested and possibly more wrong replacements. Further experiments would have to be conducted in order to evaluate the effects of such change.

## 8. Learning SHACL Relation Constraints

In this section we present our approach for translating models for the correctness of relation assertions learned with PaTyBRED into SHACL relation constraints. This approach is designed to address research question RQ2. It is important to note that we focus on the creation of constraints for relations between entities, i.e. owl:ObjectProperty. Constraints for owl:DataProperty relations containing, e.g. numerical, textual or geographical data, are out of the scope of this paper.

Learning such constraints has an important advantage when comparing to opaque relation assertion error detection methods, such as embeddings. The SHACL constraints are human-readable and can be directly evaluated and improved by specialists without requiring the manual evaluation of its output. Furthermore, once learned, they can be deployed in the knowledge graph creation process and evaluated more efficiently.

### *8.1. SHACL*

Shapes Constraint Language (SHACL) is a language for validating RDF graphs against a set of conditions, which are provided as *shapes* expressed in the form of an RDF graph called *shapes graph*. The RDF graphs that are validated against a shapes graph are called *data graphs*. The shape graphs conditions may be used for a variety of purposes beside validation, including user interface building, code generation and data integration. SHACL was created as an extension of ShEX (Shape Expressions).[17]

The SHACL specification is divided into SHACL Core and SHACL-SPARQL.[18] SHACL Core consists of frequently needed features for the representation of shapes, constraints and targets. The SHACL Core language defines shapes about the focus node itself (node shapes) and shapes about the values of a particular property or path for the focus node (property shapes).

SHACL-SPARQL consists of all features of SHACL Core plus the advanced features of SPARQL-based constraints and an extension mechanism to declare new constraint components. Constraint can be written as SPARQL ASK or SELECT queries. These queries are interpreted against each shape focus node. If an ASK query does not evaluate to true for a given node, then the constraint is violated. Constraints described using a SELECT query must return an empty result set when conforming with the constraint and non-empty set when violated.

SHACL also supports three different constraint severity levels: Info, Warning and Violation. The different levels have no impact on the validation, but may be used by to categorize validation results. It is up to the user to define how the different severity levels are handled.

### *8.2. Generation Process*

To generate SHACL constraints, we follow the idea of generating rules from decision trees. Hence, we first run PaTyBRED with a tree learner to generate a decision tree for classifying assertions into correct and erroneous ones, and extract rules for erroneous statements. Those rules are then expressed as SHACL constraints. Following [56], the trees are not optimized or pruned during learning, but we apply a specific pruning procedure later in the process.

To create the constraints, we consider the subtrees whose leave nodes state that the example should be classified as erroneous. The subtree is then converted it into a logical expression, whose negation is used as a constraint for the relation. The idea is that we used as constraints the negation of the expression that defines the examples which are predicted by PaTyBRED to be highly erroneous. In the rest of this section we describe in details how the generation of the constraints is done.

[17] https://www.w3.org/2001/sw/wiki/ShEx
[18] https://www.w3.org/TR/shacl/

Firstly we identify the nodes which contain only – or mostly – erroneous relation assertions. For a node not to be pruned it needs to satisfy minimum support and confidence thresholds, or be an ancestor of a node which satisfies the thresholds. If a non-leaf node satisfies both thresholds, all its ancestors can be pruned (to avoid redundancies). This pruned tree can then be directly converted into a logical expression which will translate the conditions into a single SHACL constraint. Each literal $L_{i,j}$ is a variable which may be negated or not. This can be directly translated to node conditions in the tree which are satisfied (right branch) or not (left branch).

Figure 7 shows an example of how the pruning process works. The decision tree is learned on the example relation `relation` from the introductory example. Leaves which contain only negative examples (like the upper right leaf) or have an impure distribution (like the lower right leaf) are pruned. From the remaining paths in the tree, logic expressions for valid relation assertions are generated.

A confidence value of 1 means that only pure nodes containing exclusively negative examples can be selected. It also means that if the learned constraints are to be applied on the original data, no existing errors can be detected. In order to enable detection of preexisting errors, the confidence threshold of less than 1 is necessary. We can use different confidence thresholds to define different SHACL constraints with different severity levels. Constraints with lower confidence may be used as warnings, while higher confidence values close to 1 maybe used as violations.

Since PaTyBRED relies on path and type features, all conditions in the decision tree nodes will be of the following kinds: subject type, object type and path.

The decision tree's logical expression can be directly translated to SHACL Core using `sh:and`, `sh:or` and `sh:not`. A shape for a relation `:r` can be defined with `:rShape a sh:NodeShape`. We define the target nodes of the shape as subjects of the target relation with `sh:targetSubjectsOf`. Subject type features test if the subject of the relation assertion is of a certain class `:C`. This can be done in SHACL with `:rShape sh:class :C`. Moreover, the object can be restricted to a type `:C` with the following expression `:rShape sh:property [sh:path :r; sh:class :C]`.

The main problem with SHACL Core is when translating path features. In the decision trees we consider pairs of subject and object as examples, however SHACL validation is performed on single nodes basis. Its vocabulary provides the components for property pair constraints `sh:equals` and `sh:disjoint`. The first requires that for all focus nodes the set of nodes reach by both properties (or property paths) should be identical, while the second requires that the sets are disjoint. The problem is that what we need to represent is the subsumption relation between a pairs of paths.

This can be illustrated with Example 1. If we want to validate the relation `:playedFor` we need to consider the subject-object pairs (`:Anelka`, `:Chelsea`) and (`:Anelka`, `:Arsenal`). Assuming every $(s, o)$ pair is required to also be connected by the path `:livedIn/:^locatedIn` in order to be correct, then both assertions should be valid. However, since the set of objects reached from `:Anelka` with `:playedFor` is `{:Chelsea, :Arsenal}` and with `:livedIn/:^locatedIn` is `{:Chelsea, :Arsenal, :Westham}`, an error on the focus node `:Anelka` would be detected if we use `sh:equals` to represent the path pattern.

Example 1:

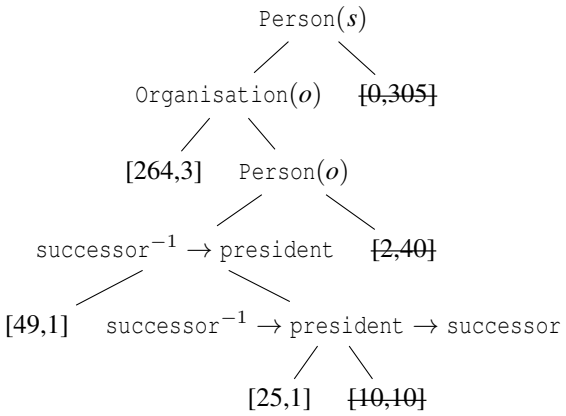
```
:Anelka    :playedFor  :Chelsea .
:Anelka    :playedFor  :Arsenal .
:Anelka    :livedIn    :London .
:Chelsea   :locatedIn  :London .
:Arsenal   :locatedIn  :London .
:Westham   :locatedIn  :London .
```

Example 2:

```
:Anelka    :playedFor  :Chelsea .
:Anelka    :playedFor  :Arsenal_ARG .
:Anelka    :livedIn    :London .
:Chelsea   :locatedIn  :London .
:Arsenal_ARG :locatedIn  :Sarandi .
:Westham   :locatedIn  :London .
```

A similar problem happens if we try to use the negation of `sh:disjoint`. In Example 2 the pair (`:Anelka`, `:Chelsea`) is correct, while (`:Anelka`, `:Arsenal_Sarandi`) is incorrect, since the pair is not connected with `:livedIn/:^locatedIn` because `:Anelka` did not live in `:Sarandi`. If we validate the data using the negation of `sh:disjoint`, the sets of objects reached with the two paths are not disjoint because both have `:Chelsea`, therefore the validator would assume that for the focus node `:Anelka` there is no assertion error with relation `:playedFor`. This would only work if the relation `:playedFor` were functional. For that reason, we cannot correctly translate the PaTyBRED decision trees into SHACL Core.

In SHACL-SPARQL path features can be correctly translated in a more intuitive way, since it is possible to work directly with subject-object pairs.




$$\begin{aligned}
&\texttt{Person}(s) \wedge \texttt{Organisation}(o) \\
\vee\ &\texttt{Person}(s) \wedge \texttt{Person}(o) \wedge (\texttt{successor}^{-1} \circ \texttt{president})(s,o) \\
\vee\ &\texttt{Person}(s) \wedge \texttt{Person}(o) \wedge (\texttt{successor}^{-1} \circ \texttt{president} \circ \texttt{successor})(s,o)
\end{aligned}$$

Fig. 7. Deriving constraints from a learned decision tree. First, leaves are pruned (marked as struck through). Then, logical constraints are derived from the remaining paths in the tree (lower part).

Moreover, it has the advantage of using a well-established and widely used language instead of requiring the learning of a whole new vocabulary. The template for a SHACL-SPARQL relation constraint is shown below. The SPARQL constraint is defined with the `sh:SPARQLConstraint` component. The variable `$this` indicate the focus node and `?o` its correspondent objects in the target relation.

```
:relSHACLShape a sh:NodeShape ;
sh:targetSubjectsOf :rel ;
sh:sparql [
 a sh:SPARQLConstraint ;
 sh:select """
  SELECT $this ?o
  WHERE {
   $this :rel ?o .
   FILTER(!(E))
  }
 """ ;
] .
```

The relation constraints expression is represented by `E`, which is negated because during validation the select query needs to return an empty set if `$this` satisfies the constraint. Table 8 shows how the PaTyBRED features can be converted into SHACL-SPARQL and Core. The path `:p` represents a property chain `:r1/.../:rn` in SHACL-SPARQL, with the `^` character before a relation indicating the inverse of the relation.

For the earlier `president` relation example from DBpedia, which corresponds to the decision tree shown in Fig. 7, the expression `E` could be defined as shown below. Every variable in the logical formula is expressed as a different EXISTS clause. Negated literals can be represented by simply negating a single variable EXISTS clause. Alternatively, disjunctions and conjunctions can be represented in a single EXISTS clause using "UNION" and "." respectively, however expressing negations would be complicated.

```
EXISTS {?o a :Person} &&
(EXISTS {$this a :Organisation} ||
 (EXISTS {$this a :Person} &&
  (EXISTS {
   $this ^:successor/:president ?o
  }
   ||
  EXISTS {
   $this ^:successor/:president/:successor ?o
  }
 )
)
)
```

It is important to note that the number of variables and the length of the expression will depend on the number of features selected defined by PaTyBRED. It also depends on the decision tree settings, such as the maximum depth, maximum number of leaf nodes, minimum samples on leaf and on split.

| Feature | SHACL-SPARQL | SHACL Core |
|---|---|---|
| $C(s)$ | `{$this a :C}` | `_:b sh:class :C .` |
| $C(o)$ | `{?o a :C}` | `_:b sh:property [sh:path :r; sh:class :C] .` |
| $p(s,o)$ | `{$this :p ?o}` | N/A |
| $p(X,s)$ | `{?X :p $this}` | `_:b sh:property [sh:path [sh:inversePath :p]] .` |
| $p(s,X)$ | `{$this :p ?X}` | `_:b sh:property [sh:path :p] .` |
| $p(X,o)$ | `{?X :p ?o}` | N/A |
| $p(o,X)$ | `{?o :p ?X}` | N/A |

Table 8

PaTyBRED features translation into SHACL

## 9. Relation Constraint Experiments

To evaluate the learning of relation constraints, we compare the constraints learned with our approach with domain and range restriction axioms learned with statistical schema induction (SSI). We conduct experiments on two large-scale knowledge graphs, i.e., DBpedia and YAGO. These experiments address research question RQ2.

As discussed above, approaches learning explicit interpretable and executable models for identifying errors in knowledge graphs are scarce, since most approaches are rather focused towards scoring individual triples. However, a feasible way of combining error detection in knowledge graph with learning explicit models is first to enrich the underlying schema or ontology by additional axioms, and then to use the axioms to detect errors in the knowledge graph [63]. We use an approach called *Statistical Schema Induction (SSI)* first introduced in [66], which uses association rule mining to learn domain and range restrictions in a schema. SSI [66] uses association rule mining to induce domain and range restrictions the data. In order to learn such restrictions, it generates transaction tables where transactions correspond to relation assertions and items correspond to relation and subject types, for domain learning, or relation and object types, for range learning. Then rules of the forms $\exists r.\top \sqsubseteq C$ and $\exists r^{-1}.\top \sqsubseteq C$ (i.e., domain and range axioms respectively) are learned with association rule mining. To compare these domain and range restrictions to our SHACL constraints, we converted them to explicit tests, flagging axioms with a given property but the subject or object missing the type defined as domain or range, respectively.

The reasons why we choose SSI as a comparison is two-fold: first, it scales well to an entire knowledge graph such as DBpedia. Second, our approach can, as discussed in the introduction, learn more complex patterns for errors which go beyond simple domain and range restrictions. Hence, the comparison will also reveal whether this theoretical capability is also exploited in practice, or whether our approach falls back to learn simple domain and range restrictions, which are only *expressed* by more complex SHACL constraints.

We run both methods with minimum confidence of 0.95 and minimum support of 50 instances. For SSI, we use the most specific domain and range axioms that satisfy the minimum confidence and support thresholds. Every constraint and axiom preserves its original confidence value, and for every fact violating the constraints we assign the confidence of its original axiom.

We rank the detected errors by the scores, and select the top-10000 (top-10k) errors with each method (less than 1% of the total amount of relation assertions). Since many of triples are in the top-10k of both methods, we manually evaluate only those triples which are selected by one method and not the other.

We decided to evaluate the compared approaches based on their ability to detect existing errors. Evaluating the quality of the generated constraints by themselves, without considering their ability to detect errors, would be subjective. Since both methods induce the constraints from the ABox and the detection of errors is their main application, we think it is fair to evaluate the approaches by how accurately they can detect errors in an incorrect dataset like DBpedia.

The learned SHACL constraints are translated from PaTyBRED decision trees learned with $mpl = 2$, $mppl = 5000$, $k = 10$ and $nneg = 1$. Out of 646 `owl:ObjectProperty` relations from DBpedia 2015-10 considered, we learned 440 SHACL constraints. Out of those 122 were simple domain and range restrictions, 224 were combinations of subject and object types and 94 had path features (from which 43 had length 2). The relevance of triangular path features in DBpedia

is rather small, contributing to only 6% of the features selected (c.f. Table 1).

Figure 8 shows the results of our manual evaluation on DBpedia[19]. Since there is some overlap in the top-10k triples detected with each method (380 triples in DBpedia and 5963 in YAGO), we also present the results of the evaluation on the differences between the two methods in Figure 9. We call SHACL-SSI the set of triples selected by the SHACL constraints and not by SSI, and SSI-SHACL the set of those selected by SSI and not SHACL. We then select random samples of 100 errors from SHACL-SSI and SSI-SHACL and manually evaluate them.

In the manual evaluation we classify the triples detected as errors into four categories.

- WT-CC: wrong triple with correct types
- WT-WC: wrong triple with wrong types
- CT-WC: correct triple with wrong types
- CT-CC: correct triple with correct types

We consider a fact to have wrong type (WC), if either the subject or the object in the triple has wrong or missing triple assertions. That includes instances which are untyped, has too general types, or has wrong type assertions. A relation assertion is considered correct (CT) if the pair of subject and object entities is correct, independent of their types.

The results from Figure 5 show that the SHACL constraints are better at detecting wrong triples, with a higher number of wrong triples with correct types (WT-CC), which are more difficult to detect. Also, the number of correct triples with wrong types (CT-WC) is reduced, showing that the more flexible SHACL constraints are better at modeling noisy and incomplete relations. We suppose that on datasets where path features are more relevant, our learned SPARQL constraints would have a greater advantage when compared to SSI, since the latter only exploits subject and object types.

We illustrate the results obtained with our method showing two examples of SHACL constraints learned on DBpedia learned for the relations `parent` and `kingdom`, as well as the relation `isCitizenOf` learned on YAGO. The `:parentShape` constraint uses exclusively path features, and it exploits the fact that generally people have children with their spouses and that it is the inverse of the child relations. In the DBpedia ontology child and parent are not the inverse of each

[19]The manual annotations can be accessed in http://data.dws.informatik.uni-mannheim.de/hmctp/shacl-eval/

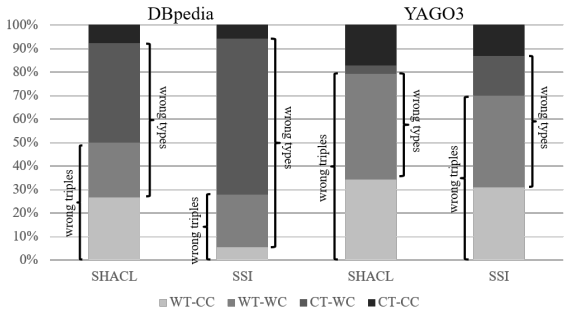



Fig. 8. Manual evaluation on DBpedia and YAGO

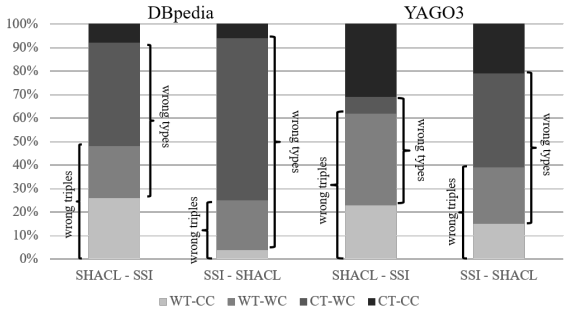



Fig. 9. Manual evaluation of the differences between SHACL and SSS on DBpedia and YAGO

other, with the two relations having different number of assertions. By considering the two path features, the constraint is more flexible requiring that neither paths connect subject and object for a relation assertion to violate the constraint. Such flexibility is particularly important on incomplete datasets, such as DBpedia.

```
:parentShape a sh:NodeShape ;
sh:targetSubjectsOf :parent ;
sh:sparql [
 a sh:SPARQLConstraint ;
 sh:select """
  SELECT $this ?o WHERE {
    $this :parent ?o .
    FILTER(
     !EXISTS {$this :parent/:spouse ?o}
      &&
     !EXISTS {$this ^:child ?o}
    )}
 """ ;
] .

:kingdomShape a sh:NodeShape ;
sh:targetSubjectsOf :kingdom ;
sh:sparql [
 a sh:SPARQLConstraint ;
 sh:select """
```

```
  SELECT $this ?o WHERE {
   $this :kingdom ?o .
   FILTER(
    !EXISTS {$this :family/:kingdom ?o}
     &&
    !EXISTS {$this :phylum/:kingdom ?o}
     &&
    !EXISTS {$this :genus/:kingdom ?o}
   )}
  """ ;
] .

:isCitizenOfShape a sh:NodeShape ;
sh:targetSubjectsOf :isCitizenOf ;
sh:sparql [
  a sh:SPARQLConstraint;
  sh:select """
   SELECT $this ?o WHERE {
    $this :isCitizenOf ?o .
    FILTER(
    !EXISTS {?o a :Country} ||
     (!EXISTS
      {$this :wasBornIn/:isLocatedIn ?o}
       &&
     !EXISTS
      {$this :graduatedFrom/:isLocatedIn ?o}
    ))}
  """ ;
] .
```

The `:isCitizenOfShape` constraint learned on YAGO3 requires that the object of the relation is of the type `:Country` and that the subject was born in a place located in the country or graduated from an institution located in the country. Although the constraint is not entirely correct, since people who were not born in or did not graduate in a country can still be citizen of a country, however, it reveals interesting patterns in the data. Moreover, by varying the minimum confidence threshold one can obtain more aggressive constraints, such as the one shown above, or more conservative ones which do not require the paths conditions to be fulfilled.

The `:kingdomShape` exploits the fact that for every level of the life taxonomy below kingdom (from species to phylum), most instances have assertions of the kingdom relation. The constraint requires that for every pair of subject-object at least one of following three paths should exist: `:family/:kingdom`, `:phylum/:kingdom ?o` and `:genus/:kingdom`. The problem is that while this holds for the majority of the `:kingdom` assertions, those which have a phylum as subject cannot have one of the three aforementioned paths because phylum is the level directly kingdom. Statistical methods – including our approach – identifies such case as outlier, since the proportion of subjects which are phyla is very small. This happens because there are orders of magnitude more species, genera, families, orders and classes than phyla.

This case illustrate the importance of having readable constraints, which can be understood and improved by specialists. The constraint could be easily fixed by adding the path `^:phylum/:kingdom` to the expression, which would include the cases where the subject is a phylum into the definition.

### *9.1. Limitations*

One of the limitations of our approach is the cost of considering paths of length $mpl > 2$ on datasets with many relations. In order to enable PaTyBRED to be used on large-scale datasets, such as DBpedia and NELL, conservative values for *mpl* and *mppl* need to be selected. This reduces the number of paths whose adjacency matrix needs to be computed and the number of features considered in the relations' training data. This improves the scalability, however, it also means that possibly relevant paths can be left out.

Another limitation is that in its current implementation, PaTyBRED generates negative examples by substituting the subject or object by a randomly selected entity. Since the distribution of instances over classes on most KGs is highly skewed, with some classes being much more likely to be sampled than others. That means the generation of potentially relevant negative examples with instances of infrequent classes is unlikely, which may make it difficult to learn constraints with such infrequent classes.

In order to compensate for this effect, we would need to introduce a bias to selection of entities on the generation of negative examples. A possible solution is to make it more likely to generate instances of the same or sibling classes, making it more likely to select entities of classes that are more closely related to the class of the original entity. That is an interesting problem, however it requires extensive research in order to verify its effectiveness on mitigating the issue.

## 10. Conclusion and Future Work

In this paper, we have investigated three research questions: error detection in knowledge graphs (RQ1), developing a method for sustaining the results of error detection and abstract from individual errors detected to patterns of such errors (RQ2), and automatic correction of such errors (RQ3).

We have shown that although the error detection problem is similar to knowledge completion, methods which perform well in knowledge completion might not necessarily be appropriate for error detection. To address RQ1, we have proposed PaTyBRED, a robust supervised error detection method which relies on type and path features, and compare it with state-of-the-art error detection and knowledge graph completion methods. We demonstrate the importance of combining those path and type features together, and we also perform a manual evaluation of our approach on DBpedia and NELL.

The experiments in our paper show that path features are particularly helpful when detecting the less obvious kinds of errors, e.g., when two entities of the same type are confused. At the same time, the search space for optimal path features is very large, so that a big potential of improvement lies in the development of efficient searching and pruning strategies. For example, in our paper, we have imposed a fixed number of paths of each length to inspect and to create longer paths from, while a flexible approach which always inspects a different fraction of paths of each length might yield better results.

To address RQ3, we have presented CoCKG, an approach for correcting erroneous facts originated from entity confusions in knowledge graphs. The experiments show that CoCKG is capable of correcting wrong triples with confused instances, with estimated precision of 21% of the produced corrections in DBpedia and 14% in NELL. The low precision values obtained do not allow this process, as of now, to be used for fully automatic KG enrichment. Nevertheless, it works as a proof of concept and can be useful, e.g., as suggestions from which a user would ultimately decide whether to execute. Moreover, fusing multiple external signals (e.g., confidence scores of link prediction approaches, external evidence from texts [23, 24], other knowledge graphs [7] or fact validation engines [29]) to achieve better scores for the substitution candidates might be a way to improve the performance of CoCKG.

We have observed that there are quite a few characteristic patterns of confusion in knowledge graphs (e.g., artists and albums with the same name, a city and a sports club located in that city, etc.). Similar to learning patterns for typical shapes in a knowledge graph, it might be interesting to learn typical shapes for confusions. Those may serve as good starting points for semi-automatically curating editing guidelines with common mistakes and how to avoid them.

To address RQ2, we have furthermore proposed a method for learning SHACL-SPARQL constraints for relations which is based on the relation assertion error detection method PaTyBRED. We compare the learned SHACL constraints with RDFS domain and range restriction learned with statistical schema induction. We performed a manual comparison of the two approaches on DBpedia, and we show that our SHACL constraints are better at detecting wrong relation assertions while being more robust when handling noise and incompleteness of subject and object type assertions. The SHACL constraints learned are available online[20] and could be deployed directly for error detection on DBpedia. These results show that, if using symbolic learning for error detection in knowledge graphs, it is possible to generate an executable model for error detection in knowledge graphs. Such an approach has two advantages: (1) manual validation with a human in the loop becomes easier when only a small number of constraints has to be reviewed instead of a large number of flagged triples, and (2) there are tools to validate RDF graphs using SHACL [19], which are used in the pipelines of building large-scale knowledge graphs. Hence, the results of error detection can be made available in a reusable way and built into the knowledge graph construction process.

In the future we plan to investigate the creation of SHACL constraints for numerical and textual data. For numerical data constraints we can extend previous works [16, 42] on the area to derive intervals which can be used as constraints. It would also be interesting to adapt CoCKG to support active learning. Since guaranteeing the quality of the newly generated facts is crucial, having input from the user to clarify borderline cases and improve the overall results would be highly valuable. Furthermore, using an ensemble of different KG models with different characteristics, e.g. KG embeddings, instead of a single model may potentially increase the robustness of the system. Finally, it would be worth adding textual features from entities descriptions to help determine if a pair of entities is related or not.

## Acknowledgements

The work presented in this paper has been partly supported by the Ministry of Science, Research and

[20] https://github.com/aolimelo/kged

the Arts Baden-Württemberg in the project SyKo$^2$W$^2$ (Synthesis of Completion and Correction of Knowledge Graphs on the Web).

## References


[1] Arndt, D., De Meester, B., Dimou, A., Verborgh, R., Mannens, E.: Using rule-based reasoning for RDF validation. In: Costantini, S., Franconi, E., Van Woensel, W., Kontchakov, R., Sadri, F., Roman, D. (eds.) Proceedings of the International Joint Conference on Rules and Reasoning. Lecture Notes in Computer Science, vol. 10364, pp. 22–36. Springer (Jul 2017)

[2] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference. pp. 722–735. ISWC'07/ASWC'07, Springer-Verlag, Berlin, Heidelberg (2007), http://dl.acm.org/citation.cfm?id=1785162.1785216

[3] Bordes, A., Glorot, X., Weston, J., Bengio, Y.: Joint learning of words and meaning representations for open-text semantic parsing. In: Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, April 21-23, 2012. pp. 127–135 (2012), http://jmlr.csail.mit.edu/proceedings/papers/v22/bordes12.html

[4] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 26, pp. 2787–2795. Curran Associates, Inc. (2013), http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf

[5] Bordes, A., Weston, J., Collobert, R., Bengio, Y.: Learning structured embeddings of knowledge bases. In: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011 (2011), http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3659

[6] Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (Oct 2001), http://dx.doi.org/10.1023/A:1010933404324

[7] Bryl, V., Bizer, C.: Learning conflict resolution strategies for cross-language Wikipedia data fusion. In: 23rd International Conference on World Wide Web. pp. 1129–1134. ACM (2014)

[8] Bühmann, L., Lehmann, J.: Universal owl axiom enrichment for large knowledge bases. In: Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management. pp. 57–71. EKAW'12, Springer-Verlag, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-33876-2_8

[9] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E.H., Mitchell, T.: Toward an architecture for never-ending language learning. In: Proceedings of the Conference on Artificial Intelligence (AAAI). pp. 1306–1313. AAAI Press (2010)

[10] Chang, K.W., Yih, S.W.t., Yang, B., Meek, C.: Typed tensor decomposition of knowledge bases for relation extraction. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. ACL âĂŞ Association for Computational Linguistics (October 2014), https://www.microsoft.com/en-us/research/publication/typed-tensor-decomposition-of-knowledge-bases-for-relation-extraction/

[11] Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global RDF vector space embeddings. In: International Semantic Web Conference (1). Lecture Notes in Computer Science, vol. 10587, pp. 190–207. Springer (2017)

[12] Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. 20(3), 273–297 (Sep 1995), http://dx.doi.org/10.1023/A:1022627411411

[13] Debattista, J., Lange, C., Auer, S.: A preliminary investigation towards improving linked data quality using distance-based outlier detection. In: Semantic Technology - 6th Joint International Conference, JIST 2016, Singapore, Singapore, November 2-4, 2016, Revised Selected Papers. pp. 116–124 (2016), https://doi.org/10.1007/978-3-319-50112-3_9

[14] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine learning research 7(Jan), 1–30 (2006)

[15] Färber, M., Bartscherer, F., Menne, C., Rettinger, A.: Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. Semantic Web 9(1), 77–129 (2018)

[16] Fleischhacker, D., Paulheim, H., Bryl, V., Völker, J., Bizer, C.: Detecting errors in numerical linked data using cross-checked outlier detection. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) The Semantic Web – ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I. pp. 357–372. Springer International Publishing, Cham (2014), https://doi.org/10.1007/978-3-319-11964-9_23

[17] Galárraga, L.A., Teflioudi, C., Hose, K., Suchanek, F.M.: AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In: Schwabe, D., Almeida, V.A.F., Glaser, H., Baeza-Yates, R.A., Moon, S.B. (eds.) 22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013. pp. 413–422. International World Wide Web Conferences Steering Committee / ACM (2013), http://dl.acm.org/citation.cfm?id=2488425

[18] Gardner, M., Mitchell, T.M.: Efficient and expressive knowledge base completion using subgraph feature extraction. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015. pp. 1488–1498 (2015), http://aclweb.org/anthology/D/D15/D15-1173.pdf

[19] Gayo, J.E.L., Prud'Hommeaux, E., Boneva, I., Kontokostas, D.: Validating rdf data. Synthesis Lectures on Semantic Web: Theory and Technology 7(1), 1–328 (2017)

[20] Gayo, J.E.L., Prud'hommeaux, E., Solbrig, H.R., Boneva, I.: Validating and describing linked data portals using shapes. CoRR abs/1701.08924 (2017), http://arxiv.org/abs/1701.08924

[21] Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: Owl 2: The next step for owl. Web Semantics: Science, Services and Agents on the World Wide Web 6(4), 309–322 (2008)

[22] Han, X., Cao, S., Lv, X., Lin, Y., Liu, Z., Sun, M., Li, J.: Openke: An open toolkit for knowledge embedding. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 139–144 (2018)

[23] Heist, N., Hertling, S., Paulheim, H.: Language-agnostic relation extraction from abstracts in wikis. Information 9(4), 75 (2018)

[24] Heist, N., Paulheim, H.: Language-agnostic relation extraction from Wikipedia abstracts. In: International Semantic Web Conference. pp. 383–399. Springer (2017)

[25] Jenatton, R., Roux, N.L., Bordes, A., Obozinski, G.R.: A latent factor model for highly multi-relational data. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 3167–3175. Curran Associates, Inc. (2012), http://papers.nips.cc/paper/4744-a-latent-factor-model-for-highly-multi-relational-data.pdf

[26] Kadlec, R., Bajgar, O., Kleindienst, J.: Knowledge base completion: Baselines strike back. CoRR abs/1705.10744 (2017), http://arxiv.org/abs/1705.10744

[27] Lao, N., Cohen, W.W.: Relational retrieval using a combination of path-constrained random walks. Mach. Learn. 81(1), 53–67 (Oct 2010), http://dx.doi.org/10.1007/s10994-010-5205-8

[28] Lao, N., Mitchell, T., Cohen, W.W.: Random walk inference and learning in a large scale knowledge base. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. pp. 529–539. EMNLP '11, Association for Computational Linguistics, Stroudsburg, PA, USA (2011), http://dl.acm.org/citation.cfm?id=2145432.2145494

[29] Lehmann, J., Gerber, D., Morsey, M., Ngomo, A.C.N.: Defacto-deep fact validation. In: International semantic web conference. pp. 312–327. Springer (2012)

[30] Lehmann, J., Hitzler, P.: A refinement operator based learning algorithm for the *ALC* description logic. In: ILP. Lecture Notes in Computer Science, vol. 4894, pp. 147–160. Springer (2007)

[31] Lehmann, J., Voelker, J.: An introduction to ontology learning. In: Lehmann, J., Voelker, J. (eds.) Perspectives on Ontology Learning. pp. ix–xvi. AKA / IOS Press (2014), http://jens-lehmann.org/files/2014/pol_introduction.pdf

[32] Lin, Y., Liu, Z., Sun, M.: Modeling relation paths for representation learning of knowledge bases. CoRR abs/1506.00379 (2015), http://arxiv.org/abs/1506.00379

[33] Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. pp. 2181–2187. AAAI'15, AAAI Press (2015), http://dl.acm.org/citation.cfm?id=2886521.2886624

[34] Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on. pp. 413–422. IEEE (2008)

[35] Mahdisoltani, F., Biega, J., Suchanek, F.M.: Yago3: A knowledge base from multilingual wikipedias (2015)

[36] 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. In: International Semantic Web Conference. pp. 3–20. Springer (2018)

[37] Melo, A., Paulheim, H.: An approach to correction of erroneous links in knowledge graphs. In: Quality Engineering Meets Knowledge Graph : QEKGraph Workshop co-located with the International Conference on Knowledge Capture (K-CAP 2017), Austin, TX, United States, December 4, 2017. pp. 1–4. ACM, New York, NY (2017), http://ub-madoc.bib.uni-mannheim.de/43852/

[38] Melo, A., Paulheim, H.: Detection of relation assertion errors in knowledge graphs. In: Corcho, Ó., Janowicz, K., Rizzo, G., Tiddi, I., Garijo, D. (eds.) Proceedings of the Knowledge Capture Conference, K-CAP 2017, Austin, TX, USA, December 4-6, 2017. pp. 22:1–22:8. ACM (2017), http://doi.acm.org/10.1145/3148011.3148033

[39] Melo, A., Paulheim, H.: Local and global feature selection for multilabel classification with binary relevance. Artificial intelligence review (2017), http://ub-madoc.bib.uni-mannheim.de/42213/

[40] Melo, A., Paulheim, H.: Synthesizing knowledge graphs for link and type prediction benchmarking. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) The Semantic Web. pp. 136–151. Springer International Publishing, Cham (2017)

[41] Melo, A., Paulheim, H., Völker, J.: Type prediction in rdf knowledge bases using hierarchical multilabel classification. In: Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics. pp. 14:1–14:10. WIMS '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2912845.2912861

[42] Melo, A., Theobald, M., Völker, J.: Correlation-based refinement of rules with numerical attributes. In: Proceedings of the twenty-seventh International Conference of the Florida Artificial Intelligence Research Society (FLAIRS) : May 21 - 23, 2014 Pensacola Beach, Florida, USA. pp. 345–350. AAAI Press, Palo Alto, Calif. (2014), http://ub-madoc.bib.uni-mannheim.de/35956/

[43] Mihov, S., Schulz, K.U.: Fast approximate search in large dictionaries. Comput. Linguist. 30(4), 451–477 (Dec 2004), http://dx.doi.org/10.1162/0891201042544938

[44] Muñoz, E., Nickles, M.: Mining cardinalities from knowledge bases. In: DEXA (1). Lecture Notes in Computer Science, vol. 10438, pp. 447–462. Springer (2017)

[45] Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. 33(1), 31–88 (Mar 2001), http://doi.acm.org/10.1145/375360.375365

[46] Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proceedings of the IEEE 104(1), 11–33 (2016), https://doi.org/10.1109/JPROC.2015.2483592

[47] Nickel, M., Rosasco, L., Poggio, T.A.: Holographic embeddings of knowledge graphs. CoRR abs/1510.04935 (2015), http://arxiv.org/abs/1510.04935

[48] Nickel, M., Tresp, V., peter Kriegel, H.: A three-way model for collective learning on multi-relational data. In: Getoor, L., Scheffer, T. (eds.) Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 809–816. ACM, New York, NY, USA (2011), http://www.icml-2011.org/papers/438_icmlpaper.pdf

[49] Paulheim, H.: Data-driven joint debugging of the dbpedia mappings and ontology. In: European Semantic Web Conference. pp. 404–418. Springer (2017)

[50] Paulheim, H.: Knowledge graph refinement: A survey of approaches and evaluation methods. Semantic web 8(3), 489–508 (2017)

[51] Paulheim, H., Bizer, C.: Type Inference on Noisy RDF Data, pp. 510–525. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-41335-3_32

[52] Paulheim, H., Bizer, C.: Improving the quality of linked data using statistical distributions. Int. J. Semantic Web Inf. Syst. 10(2), 63–86 (2014), https://doi.org/10.4018/ijswis.2014040104

[53] Paulheim, H., Gangemi, A.: Serving DBpedia with DOLCE–More than Just Adding a Cherry on Top. In: International Semantic Web Conference. LNCS, vol. 9366. Springer, International (2015), http://dx.doi.org/10.1007/978-3-319-25007-6_11

[54] Paulheim, H., Stuckenschmidt, H.: Fast approximate a-box consistency checking using machine learning. In: International Semantic Web Conference. pp. 135–150. Springer (2016)

[55] Potoniec, J., Jakubowski, P., Lawrynowicz, A.: Swift linked data miner: Mining owl 2 el class expressions directly from on-line rdf datasets. Web Semantics: Science, Services and Agents on the World Wide Web 46(1) (2017), http://www.websemanticsjournal.org/index.php/ps/article/view/504

[56] Quinlan, J.R.: Simplifying decision trees. International journal of man-machine studies 27(3), 221–234 (1987)

[57] Ringler, D., Paulheim, H.: One knowledge graph to rule them all? analyzing the differences between dbpedia, yago, wikidata & co. In: Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). pp. 366–372. Springer (2017)

[58] Ristoski, P., Paulheim, H.: RDF2Vec: RDF Graph Embeddings for Data Mining, pp. 498–514. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-46523-4_30

[59] Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., Paulheim, H.: Rdf2vec: Rdf graph embeddings and their applications. Semantic Web 10(4), 721–752 (2019)

[60] Rudolph, S.: Acquiring Generalized Domain-Range Restrictions, pp. 32–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-540-78137-0_3

[61] Shi, B., Weninger, T.: Proje: Embedding projection for knowledge graph completion (2017), https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14279

[62] Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 26, pp. 926–934. Curran Associates, Inc. (2013), http://papers.nips.cc/paper/5028-reasoning-with-neural-tensor-networks-for-knowledge-base-completion.pdf

[63] Töpper, G., Knuth, M., Sack, H.: DBpedia Ontology Enrichment for Inconsistency Detection. In: Proceedings of the 8th International Conference on Semantic Systems. pp. 33–40. ACM, New York (2012), http://dx.doi.org/10.1145/2362499.2362505

[64] Toutanova, K., Chen, D.: Observed versus latent features for knowledge base and text inference. In: 3rd Workshop on Continuous Vector Space Models and Their Compositionality. ACL âĂŞ Association for Computational Linguistics (July 2015), https://www.microsoft.com/en-us/research/publication/observed-versus-latent-features-for-knowledge-base-and-text-inference/

[65] Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. CoRR abs/1606.06357 (2016), http://arxiv.org/abs/1606.06357

[66] Völker, J., Niepert, M.: Statistical Schema Induction, pp. 124–138. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), https://doi.org/10.1007/978-3-642-21034-1_9

[67] Wang, C., Zhang, R., He, X., Zhou, A.: Error link detection and correction in wikipedia. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. pp. 307–316. CIKM '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2983323.2983705

[68] Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering PP(99), 1–1 (2017)

[69] Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: Brodley, C.E., Stone, P. (eds.) AAAI. pp. 1112–1119. AAAI Press (2014), http://dblp.uni-trier.de/db/conf/aaai/aaai2014.html#WangZFC14

[70] Weaver, G., Strickland, B., Crane, G.: Quantifying the accuracy of relational statements in wikipedia: a methodology. 2006 IEEE/ACM 6th Joint Conference on Digital Libraries 00, 358 (2006)

[71] Xiao, H., Huang, M., Hao, Y., Zhu, X.: Transg : A generative mixture model for knowledge graph embedding. CoRR abs/1509.05488 (2015), http://arxiv.org/abs/1509.05488

[72] Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Learning multi-relational semantics using neural-embedding models. CoRR abs/1411.4072 (2014), http://arxiv.org/abs/1411.4072

[73] Yang, Z., Yu, J., Kitsuregawa, M.: Fast algorithms for top-k approximate string matching. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. pp. 1467–1473. AAAI'10, AAAI Press (2010), http://dl.acm.org/citation.cfm?id=2898607.2898841