LitKGE: Improving numeric LITerals based Knowledge Graph Embedding models

Genet Asefa Gesese [a,b,*], Harald Sack [a,b] and Mehwish Alam [c]
[a] *Information Service Engineering, FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, Germany*
E-mails: genet-asefa.gesese@fiz-karlsruhe.de, harald.sack@fiz-karlsruhe.de
[b] *Institute AIFB, Karlsruhe Institute of Technology, Germany*
E-mails: genet-asefa.gesese@fiz-karlsruhe.de, harald.sack@fiz-karlsruhe.de
[c] *Télécom Paris, Institut Polytechnique de Paris, France*
E-mail: mehwish.alam@telecom-paris.fr

Abstract. Knowledge Graphs (KGs) consist of a set of triples where a triple is a link between either two entity nodes (relational triple) or an entity node and a literal node (attributive triple). Similar to relational triples, attributive triples also provide semantics that could benefit the task of link prediction (LP) on KGs. Hence, some embedding-based LP methods have been introduced in the past which leverage both kinds of triples. This paper introduces a novel approach named LitKGE where literals (specifically numerical literals) are further exploited to improve the task of LP by making implicit information that is present in the KG explicit. This is performed by generating property paths starting from an entity node leading to a literal node and considering these paths as features for the entity. The features are then incorporated with existing LP or Knowledge Graph Embedding (KGE) methods. LitKGE is evaluated on three standard LP datasets, namely FB15K-237, YAGO3-10, and LitWD48K. The results indicate that LitKGE outperforms the state-of-the-art KGE models which use numerical literals.

Keywords: Knowledge Graph Completion, Knowledge Graph Embedding, Link Prediction, Numeric Literals

1. Introduction

Knowledge Graphs (KGs) are composed of structured information to describe entities in a certain domain, and the interrelations between them. Various KGs such as DBpedia [15], Wikidata [24], and YAGO [17] have been published which play a vital role in the recent massive interest in using KGs for tasks such as question answering, search, and recommendation systems [25]. In this paper, KG is considered as a set of triples where a triple represents a link between two entity nodes or an entity node and a literal node. The former kind of triples is referred to as *relational triples* and the later as *attributive triples*. Similar to [11], this paper also uses these terms to refer to the respective triples. For example, in Figure 1, the triple <Paper_B, nominatedFor, Best_Paper_by_Emerging_Authors> is a relational triple whereas <Anna, age, 24> is an attributive triple.

As mentioned above KGs are crucial in capturing and representing facts that could be used in many applications. However, KGs are never complete due to the open-world assumption. In order to address this problem, the most common solutions perform Link Prediction (LP) which is typically referred to as the task of predicting an entity that has a specific relation with another given entity, i.e., predicting the head entity `h` given (relation `r`, tail `t`) or the tail

[*]Corresponding author. E-mail: genet-asefa.gesese@fiz-karlsruhe.de.

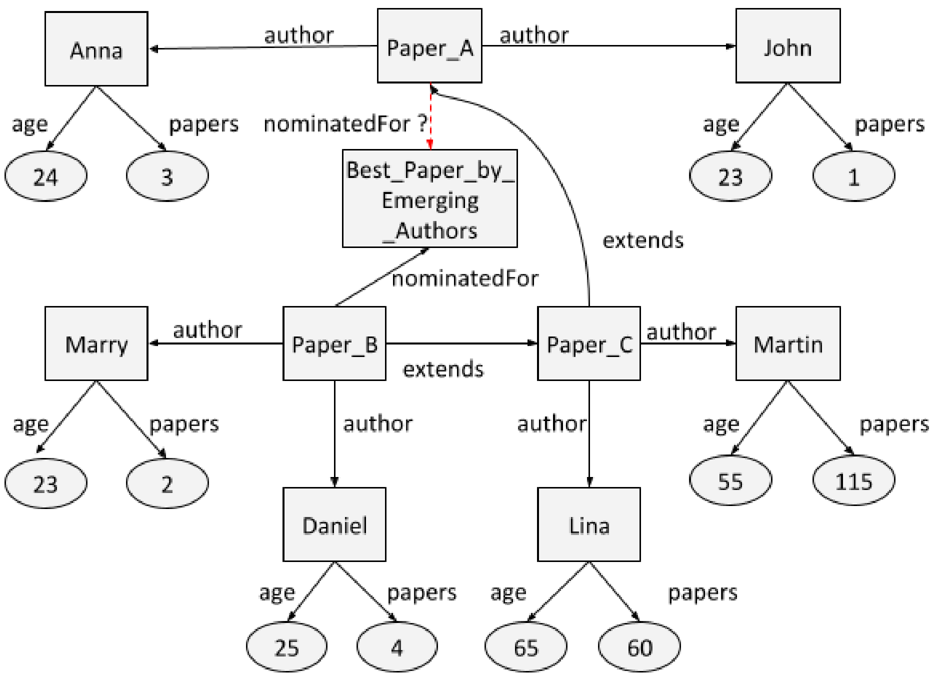

Fig. 1. An example graph with entity nodes depicted as rectangles and literals as ovals.

entity `t` given (`h`, `r`), with the former denoted as `(?, r, t)` and the latter as `(h, r, ?)` [25]. For example, `(?, DirectorOf, Wednesday)` denotes the prediction of the director of the tv-series Wednesday, while `(Tim Burton, DirectorOf, ?)` represents the prediction of films directed by Tim Burton. In general, LP methods could be categorized as embedding-based (i.e., learning latent representation for entities) or non-embedding based models. Few of the embedding-based approaches (Knowledge Graph Embedding (KGE) models) perform LP by utilizing the different kinds of literals present in the KGs [11]. One of those models is LiteralE[14] which uses both relational and attributive triples.

In this work, we investigate the benefits of making implicit information explicit to improve the task of LP. This could be achieved using the property paths leading to numerical literals as features of those entities which are indirectly associated with the literals through graph traversal. For instance, in Figure 1 given a paper (i.e., `Paper_A`, `Paper_B`, or `Paper_C`), the average age and the average number of `papers` of its authors could be used to determine if the paper can be nominated for the `Best_Paper_by_Emerging_Authors` award. Moreover, this information also has a role in providing the semantics required to determine whether `Paper_A` should be close to `Paper_B` or `Paper_C` in the vector space. This is done by generating and treating property paths that lead to literal nodes as features of entities. `author_age` and `author_papers` are among the property paths that could be generated from Figure 1. For the entity `Paper_C`, the average of the age values of Lina and Martin (i.e., 60) is taken as the value for the first feature (<Paper_C, author_age, 60>).

The generated features could be used to further enrich the attributive triples and in turn to improve the LP task. Hence, in this work, a new approach named LitKGE is proposed to enhance the performance of KGE models with literals. Furthermore, it could be integrated with many KGE models which use numerical literals such as LiteralE, TransEA [26], etc.

Note that this work is based on Chapter 7 of the first author's doctoral thesis [8] which is not peer-reviewed. It formulates and addresses the following research question: ***Does generating entity features based on property paths,***

and incorporating these features into existing KGE models result in improving LP tasks? Following are the main contributions that are achieved in this work while attempting to answer the question:

– A novel approach named LitKGE which generates features for entities in order to enhance the standard KGE models such as LiteralE, is introduced. LitKGE is a universal method which can be integrated with any KGE model which utilizes literals.
– LitKGE is evaluated on three LP datasets: FB15K-237[23], YAGO3-10[6], and LitWD48K[9]. These datasets are extended with the generated features and the extensions are made publicly available to facilitate further research[1].
– The experimental results indicate that making the implicit information explicit enabled LitKGE to outperform the State-Of-The-Art (SOTA) models.

The rest of the paper is organized as follows: Section 2 presents the SOTA KGE methods followed by a formal definition of the LP problem along with an overview of the base models considered in this work in Section 3. A detailed discussion on the proposed methodology is given in Section 4. The experiment settings and the results obtained are provided in Section 5. Finally, Section 6 summarizes the paper and points out directions for further research.

2. Related Work

Different KGE methods have been proposed so far, namely, MTKGNN [22], TransEA, KBLRN [7], and LiteralE which leverage numerical literals for the task of LP [11]. MTKGNN and TransEA utilize literals by performing the task of predicting the literal value for a given entity. On the other hand, KBLRN is a method which works by combining relational, latent, and numerical features together. The numerical features are obtained by computing the difference between the numerical literals of the head and tail entity and taken as a good predictor for a given relation.

LiteralE is a universal approach that incorporates literals into latent feature methods (KGE models) such as DistMult and ComplEx via a learnable parametric function. This function takes as input an entity embedding and a literal feature vector and maps them to a new vector of the same dimension as the entity embedding. The resulting literal enriched vector is then passed as input to the given LP scoring function. LiteralE-AT [16] extends LiteralE with more semantics from the textual labels of attributes by considering these labels as textual literals. Transforming numeric and textual literals into entities is another approach introduced recently [2]. Similarly, in [20], a set of universal preprocessing operators are proposed which can be used to transform KGs with literals for numerical, temporal, textual, and image information. In another related work [21], literals are converted into entities and used to learn embeddings for KGs specifically for the task of quality monitoring for welding in the manufacturing industry. These approaches could be applied to datasets with few triples containing numerical literal values. However, when handling millions of attributive triples, the challenge of model scalability may arise from converting all literals into entities. Furthermore, such literal-to-entity transformation can lead to skewed datasets, with many entities appearing only in the tail position.

KGA [5] is an approach that learns the representation of a hyper-relational KG containing numerical literals in either triplets or qualifiers. In another work [18], a spatial link prediction method is introduced for scenarios where the knowledge graph lacks entity relationships but contains literal values. Note that the approach is particularly tailored for spatial KGs with a limited number of entity relationships. LitKGE differs from all the above-mentioned models because it utilizes literal information not only for those entities which are directly connected to the literals but also for others which are indirectly associated with the literals. This enables LitKGE to generate numerical features for entities that do not occur in the set of attributive triples. Hence, LitKGE exploits the semantics present in the numerical literals better than the SOTA models with literals.

Generating features for entities in KGs has been investigated recently in Literal2Feature [1] for basic machine learning tasks such as regression and clustering of entities and the results achieved indicate that literals contain

[1] https://github.com/GenetAsefa/LitKGE

useful semantics about entities. In our model LitKGE, literal-based features of entities are considered as a source of semantics for the task of KG completion, specifically LP. However, the feature generation procedure of LitKGE is different from that of Literal2Feature. In Literal2Feature, a graph traversal algorithm (either Breadth-First Search or Random Walk) is applied directly to the input RDF graph in order to gather literals for each entity and consider them as a feature vector for the given entity. Then, given the generated path, it keeps only the properties and the literal value, ignoring the entities appearing between the starting entity and the literal node. Differently from Literal2Feature, in this work, LitKGE proposes a novel algorithm named `WeiDNeR_extended` which generates a weighted and directed relation-relation/attribute network, which is used as an input to a random walk algorithm to generate property paths. Then these property paths are used to collect literals to leverage them as features of entities. The `WeiDNeR_extended` algorithm enables LitKGE to efficiently generate more sound features by using the weights in the resulting network graph.

3. Preliminaries

In this section, the formal definition of the LP problem is given. Moreover, an overview of the LiteralE KGE model is provided along with a description of the scoring function of the base models DistMult and ComplEx.

3.1. Problem Definition

LP in KGs aims at predicting new relations between entities leveraging the existing triples/facts for training. Formally, let $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{D}, \mathcal{L}, \mathcal{T}\}$ be a KG where $\mathcal{E} = \{e_1, e_2, ..., e_{N_d}\}$ representing the set of entities in G, $\mathcal{R} = \{r_1, r_2, ..., r_{N_r}\}$ denotes the set of relations connecting two entities, $\mathcal{D} = \{d_1, d_2, ..., d_{N_d}\}$ as a set of attributes (a.k.a data relations) connecting entities to their corresponding literals, and $\mathcal{L}$ is the set of literal values. The triples in $\mathcal{G}$ are represented as $\mathcal{T} \subseteq ((\mathcal{E} \times \mathcal{E} \times \mathcal{R}) \cup (\mathcal{E} \times \mathcal{L} \times \mathcal{D}))$. Given $\mathcal{G}$, the task of LP can be formulated by a function $\Phi : \mathcal{E} \times \mathcal{E} \times \mathcal{R} \rightarrow \mathbb{R}$ which assigns a score to each triple $(e_i, e_j, r_k) \in \mathcal{E} \times \mathcal{E} \times \mathcal{R}$, where a higher score indicates that the triple is more likely to be valid.

3.2. LiteralE

LiteralE incorporates literals into SOTA embedding methods designed for LP such as DistMult and ComplEx. As per the experimental results presented in a survey conducted on KGE models with numeric literals [11], LiteralE outperforms all other KGE models which use literals like KBLN, MTKGNN, and TransEA. Given a base model like DistMult with a scoring function $f = f_{distmult}$ shown in Equation 1 or ComplEx with $f = f_{complex}$ in Equation 2, LiteralE modifies f by replacing the original entity vectors e_i in f with literal enriched representations e_i^{lit} using a flexible and learnable GRU-based transformation function g.

$$f_{distmult}(e_i, e_j, r_k) = e_i^T diag(r_k) e_j \tag{1}$$

$$\begin{aligned} f_{complex}(e_i, e_j, r_k) &= Re(\langle e_i, e_j, r_k \rangle) \\ &= \langle Re(e_i), Re(e_j), Re(r_k) \rangle \\ &+ \langle Im(e_i), Im(e_j), Re(r_k) \rangle \\ &+ \langle Re(e_i), Im(e_j), Im(r_k) \rangle \\ &- \langle Im(e_i), Re(e_j), Im(r_k) \rangle \end{aligned} \tag{2}$$

The function g is shown in Equation 3 which takes as an input e_i and its corresponding literal vector l_i and maps them to a new vector. Note that l_i is the i-th row of a literal matrix $\mathbf{L} \in \mathbb{R}^{N_e \times N_d}$ as the literal vector of the i-th entity. The entry L_{ik} in L is the k-th literal value of the i-th entity if an attributive triple with the i-th entity and the k-th data relation (i.e., attribute) exists in the KG, and zero otherwise.

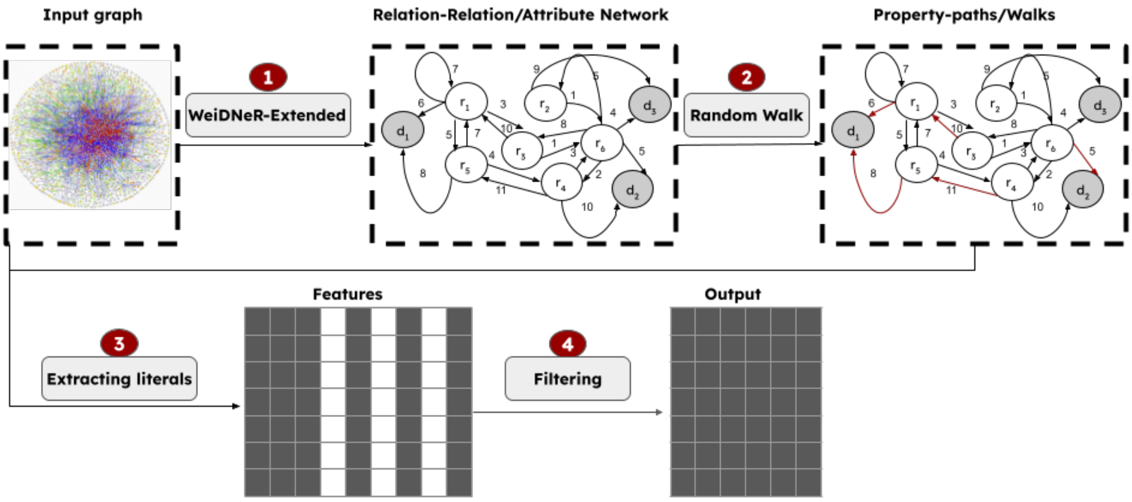

Fig. 2. Feature generation pipeline: given a KG as input, it generates a feature matrix containing numerical features for the entities in the KG.

$$\begin{aligned} g : \mathbb{R}^H \times \mathbb{R}^{N_d} &\to \mathbb{R}^H, \\ \mathbf{e}, \mathbf{l} &\mapsto \mathbf{z} \odot \mathbf{h} + (1 - \mathbf{z}) \odot \mathbf{e}, \end{aligned} \tag{3}$$

where

$$\mathbf{z} : \sigma(\mathbf{W}_{ze}^T \mathbf{e} + \mathbf{W}_{zl}^T \mathbf{l} + \mathbf{b}), \tag{4}$$

and

$$\mathbf{h} = h(\mathbf{W}_h^T [\mathbf{e}, \mathbf{l}]). \tag{5}$$

Note that $\mathbf{W}_{ze} \in \mathbb{R}^{H \times H}, \mathbf{W}_{zl} \in \mathbb{R}^{N_d \times H}, \mathbf{b} \in \mathbb{R}^H$, and $\mathbf{W}_h \in \mathbb{R}^{H+N_d \times H}$ are the parameters of g, σ is the sigmoid function, $\odot$ denotes the element-wise multiplication, and h is a component-wise nonlinearity. The scoring function $f(\mathbf{e}_i, \mathbf{e}_j, \mathbf{r}_k)$ would be replaced with $f(g(\mathbf{e}_i, \mathbf{l}_i), g(\mathbf{e}_j, \mathbf{l}_j), \mathbf{r}_k)$.

4. LitKGE

In this Section, the proposed approach LitKGE is presented in detail. It consists of two major components, i.e., i) Generating numeric features and ii) Incorporating numeric features into KGE models. These components are discussed in the subsequent sections.

4.1. Generating features

As discussed in Section 1 given a KG, implicit features of entities could be made explicit by traversing the KG using property paths. Hence, LitKGE generates such paths which lead to literal values to create features for entities. The overall phases in the feature generation process are depicted in Figure 2.

The phases are described as follows:

- **Input graph**: A KG which contains entity nodes, literal nodes, and properties (relations and attributes) is taken as input to generate numerical features.

- **WeiDNeR-Extended**: A novel algorithm named WeiDNeR-Extended is introduced which is an extension of the WeiDNeR algorithm developed in RAILD [10]. There are two major differences between WeiDNeR and WeiDNeR-Extended: i) WeiDNeR is designed based on the assumption that given a KG $\mathcal{G} = (\mathcal{R}, \mathcal{E}, \mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E})$, $r_1, r_2 \in \mathcal{R}$ and $\mathcal{T}_1 \subseteq (\mathcal{T} \cap (\mathcal{E} \times r_1 \times \mathcal{E}))$, $\mathcal{T}_2 \subseteq (\mathcal{T} \cap (\mathcal{E} \times r_2 \times \mathcal{E}))$, the higher the number of common entities between $\mathcal{T}_1$ and $\mathcal{T}_2$, the higher the probability that r_1 and r_2 could be semantically similar. In WeiDNeR-Extended, the assumption is formulated as Given a KG $\mathcal{G} = (\mathcal{R}, \mathcal{A}, \mathcal{E}, \mathcal{L}, \mathcal{T}_R \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}, \mathcal{T}_A \subseteq \mathcal{E} \times \mathcal{A} \times \mathcal{L})$, $p_1 \in \mathcal{R}$, $p_2 \in \mathcal{R} \cup \mathcal{A}$ and $\mathcal{T}_1 \subseteq (\mathcal{T}_R \cap (\mathcal{E} \times p_1 \times \mathcal{E}))$, $\mathcal{T}_2 \subseteq ((\mathcal{T}_R \cup \mathcal{T}_A) \cap (\mathcal{E} \times p_2 \times \mathcal{E} | \mathcal{L}))$, the higher the number of matches between tail entities in $\mathcal{T}_1$ and head entities in $\mathcal{T}_2$, the higher the probability that p_1 and p_2 appear close to each other in the original KG. This restriction to matching only the tail of $\mathcal{T}_1$ and the head of $\mathcal{T}_2$ is performed in order to keep only outgoing links when generating property paths. ii) As described in i), WeiDNeR generates a relation-relation network taking only relational triples as inputs, i.e., excluding literals, hence, there are no attributes in the resulting network. However, in WeiDNeR-Extended, the network is intended to contain relation-relation as well as relation-attribute connections. Algorithm 1 describes the steps followed by WeiDNeR-Extended to generate the network.
  Algorithm Description: Given two properties p_i and p_j,
  * if p_i and p_j are different relations connecting entities, it computes the weight $Weight_{<p_i,p_j>}$ by counting the number of pair of triples where the relation in the first triple is p_i and in the second is p_j and the tail entity in the first triple is the same as the head entity in the second triple. If $Weight_{<p_i,p_j>}$ is greater than zero, then p_i and p_j are considered as nodes in the output network, and a directed link from p_i to p_j is created with the computed weight. $Weight_{<p_j,p_i>}$ is computed similarly and if it is greater than 0, then a directed link from p_j to p_i is created and assigned the computed weight (i.e., refer to lines 4- 9).
  * if p_i is a relation connecting entities and p_j denotes an attribute connecting entity to a literal node, the weight $Weight_{<p_i,p_j>}$ by counting the number of pair of triples where the relation in the first triple (a relational triple) is p_i and in the second triple (an attributive triple) is p_j and the tail entity in the first triple is the same as the head entity in the second triple. If $Weight_{<p_i,p_j>}$ is greater than zero, then p_i and p_j are considered as nodes in the output network, and a directed link from p_i to p_j is created with the computed weight. The same analogy applies if p_i is an attribute and p_j is a relation so as to decide whether there should be a link from p_j to p_i and to compute the weight $Weight_{<p_j,p_i>}$ (i.e., refer to lines 11- 17).
  * if p_i and p_j are relations and they are exactly the same, then the weight $Weight_{<p_i,p_i>}$ by counting the number of pair of triples where the relation in the first triple (a relational triple) is p_i and in the second triple (an attributive triple) is p_j and the tail entity in the first triple is the same as the head entity in the second triple. If $Weight_{<p_i,p_i>}$ is greater than zero, then p_i is considered as a node in the output network, and a link from p_i to itself (i.e., a loop)is created with the computed weight.

  Note that the motivation behind creating a relation-relation/attribute network (i.e., proposing the WeiDNeR-Extended algorithm) is to make sure that within a property path that is generated as a feature, the properties are semantically related. For instance considering the example relation-relation/attribute graph in the feature generation pipeline in Figure 2, when the random walk algorithm is applied to this graph and takes $r3$ as a starting node, it selects $r1$ as the next node instead of $r6$ because the weight on the link from $r3$ to $r1$ (i.e., 10) is bigger than the weight from $r3$ to $r6$ (i.e., 1). Moreover, having the weights in such relation-relation/attribute networks allows the generation of less-sparse features. This approach is novel due to the fact that none of the existing KGE methods utilized such a technique to generate features for entities.

- **Random walk**: The 2^{nd} order-based biased random walk approach used in Node2Vec[12] is applied to generate network neighborhoods for nodes in the WeiDNeR-Extended network.

- **Extracting literals:** The generated walks/property paths with the random walk graph traversal are used as templates to obtain literals for entities in the input KG. Given an entity, some property paths could have multiple values and in such cases, their average is taken. For example, taking the graph in Figure 1 as the input KG and *author_age* as a property path generated with a random walk, for Paper_B there would be two possible literal values 23 and 25 through the entities Marry and Daniel respectively (i.e., <Paper_B, author_age, 23> and <Paper_B, author_age, 25>) and taking the average would result in <Paper_B, author_age, 24>.

- **Filtering:** In this step, those property_paths/features which have values for only one entity are removed. This is done in order to avoid having features that do not provide much semantics to help compare entities but rather make the feature matrix sparse.

- **Output**: The resulting feature matrix $\mathcal{F} \in \mathbb{R}^{N_e \times N_f}$ contains the generated numerical features for the entities in the input KG, where N_e and N_f denote the number of entities and the number of features respectively. Note that each entry $\mathcal{F}_{ik}$ contains the k-th feature value of the i-th entity if a feature value is generated for the i-th entity and the k-th feature using the feature generation pipeline, and zero otherwise.

Algorithm 1: WeiDNeR-Extended - An algorithm to generate a directed and weighted relation-relation/attribute network.

Data: $T \leftarrow Relational\ and\ Attributive\ Triples\ in\ KG$
Data: $R \leftarrow Properties\ in\ Relational\ Triples\ (i.e.,\ Relations)$
Data: $D \leftarrow Properties\ in\ Attributive\ Triples\ (i.e., Attributes)$
Result: N_{rel}

for *each pair of properties* $\langle p_i, p_j \rangle$ **do**
  if $p_i \neq p_j$ **then**
    if $p_i, p_j \in R$ **then**
      $Weight_{\langle p_i, p_j \rangle} \leftarrow \left|\{(\langle h_1, p_i, t_1 \rangle, \langle h_2, p_j, t_2 \rangle) : \langle h_1, p_i, t_1 \rangle \in T, \langle h_2, p_j, t_2 \rangle \in T, t_1 = h_2\}\right|$;
      $Weight_{\langle p_j, p_i \rangle} \leftarrow \left|\{(\langle h_1, p_i, t_1 \rangle, \langle h_2, p_j, t_2 \rangle) : \langle h_1, p_i, t_1 \rangle \in T, \langle h_2, p_j, t_2 \rangle \in T, h_1 = t_2\}\right|$;
      if $Weight_{\langle p_i, p_j \rangle} > 0$ **then**
        $N_{rel} \leftarrow N_{rel} \bigcup \{\langle p_i, p_j, Weight_{\langle p_i, p_j \rangle} \rangle\}$;
      if $Weight_{\langle p_j, p_i \rangle} > 0$ **then**
        $N_{rel} \leftarrow N_{rel} \bigcup \{\langle p_j, p_i, Weight_{\langle p_i, p_j \rangle} \rangle\}$;
    else if $p_i \in R\ \&\ p_j \in D$ **then**
      $Weight_{\langle p_i, p_j \rangle} \leftarrow \left|\{(\langle h_1, p_i, t_1 \rangle, \langle h_2, p_j, l \rangle) : \langle h_1, p_i, t_1 \rangle \in T, \langle h_2, p_j, l \rangle \in T, t_1 = h_2\}\right|$;
      if $Weight_{\langle p_i, p_j \rangle} > 0$ **then**
        $N_{rel} \leftarrow N_{rel} \bigcup \{\langle p_i, p_j, Weight_{\langle p_i, p_j \rangle} \rangle\}$;
    else
      $Weight_{\langle p_j, p_i \rangle} \leftarrow \left|\{(\langle h_1, p_j, t_1 \rangle, \langle h_2, p_i, l \rangle) : \langle h_1, p_j, t_1 \rangle \in T, \langle h_2, p_i, l \rangle \in T, t_1 = h_2\}\right|$;
      if $Weight_{\langle p_j, p_i \rangle} > 0$ **then**
        $N_{rel} \leftarrow N_{rel} \bigcup \{\langle p_j, p_i, Weight_{\langle p_j, p_i \rangle} \rangle\}$;
  else
    if $p_i \in R$ **then**
      $Weight_{\langle p_i, p_i \rangle} \leftarrow \left|\{(\langle h_1, p_i, t_1 \rangle, \langle h_2, p_j, t_2 \rangle) : \langle h_1, p_i, t_1 \rangle \in T, \langle h_2, p_j, t_2 \rangle \in T, t_1 = h_2, (h_1 \neq t_1 \vee h_1 \neq t_2)\}\right|$;
      if $Weight_{\langle p_i, p_i \rangle} > 0$ **then**
        $N_{rel} \leftarrow N_{rel} \bigcup \{p_i, p_i, Weight_{\langle p_i, p_i \rangle}\}$;

4.2. Incorporating features into KGE models

Once the features are generated using the procedure discussed in Section 4.1, they can be utilized while learning KGEs. This can be achieved by incorporating them in any KGE model which utilizes literals. For instance, TransEA and LiteralE are among those KGE models which can be improved using the LitKGE approach. TransEA is an extension of TransE model where an attribute embedding component is integrated into TransE. The attributive embedding component uses all attributive triples with numeric literals as input and applies a linear regression model to learn embeddings of entities and attributes. One way to improve TransEA with LitKGE would be to use the generated features in LitKGE and treat them as attributes and predict the values using the attributive embedding component. On the other hand, LiteralE is a universal KGE model which incorporates literals to the standard KGE models such as DistMult and ComplEx. In this work, LiteralE is selected as a base model to improve it with LitKGE due to the fact that it performs better than TransEA and other KGE models which use literals as presented in [11]. In addition to its performance, LiteralE is a universal model, i.e., as mentioned above it can be applied to many standard models, which implies that applying LitKGE to LiteralE is in turn applying it to the base scoring models.

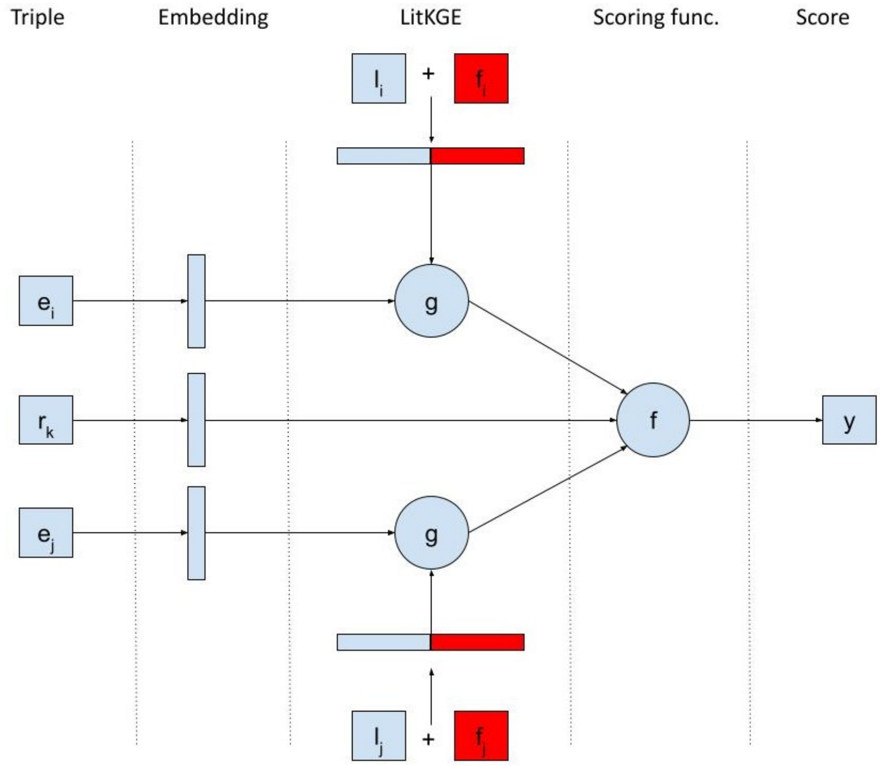

Fig. 3. Overview of LitKGE as an improvement over LiteralE model. LitKGE takes as input the embedding of the entities and the concatenation of their corresponding literal vectors l_i and feature vector f_i as input and combines them via a learnable function g (i.e., g_{lf} in Equation 6). Then, it modifies the base scoring function f with the joint embedding obtained using g.

The modification to the LiteralE architecture with LitKGE is depicted in Figure 3. Let $\boldsymbol{L} \in \mathbb{R}^{N_e \times N_d}$ be a matrix representing literals and $\boldsymbol{F} \in \mathbb{R}^{N_e \times N_f}$ be a newly introduced feature matrix generated using the procedure presented in Section 4.1. The final new matrix would be a combination of L and F as $L_F = [L; F]$, $\boldsymbol{L_F} \in \mathbb{R}^{N_e \times N_d + N_f}$ Then, the mapping function used in LiteralE (i.e., g in Equation 3) is modified as g_{lf} in Equation 6.

$$\begin{aligned} g_{lf} &: \mathbb{R}^H \times \mathbb{R}^{N_d + N_f} \to \mathbb{R}^H, \\ \boldsymbol{e}, \boldsymbol{l_F} &\mapsto \boldsymbol{z} \odot \boldsymbol{h} + (1 - \boldsymbol{z}) \odot \boldsymbol{e}, \end{aligned} \tag{6}$$

where $\odot$ is the pointwise multiplication and

$$\boldsymbol{z} : \sigma(\boldsymbol{W}_{ze}^T \boldsymbol{e} + \boldsymbol{W}_{zl}^T \boldsymbol{l_F} + \boldsymbol{b}), \tag{7}$$

and

$$\boldsymbol{h} = h(\boldsymbol{W}_h^T[\boldsymbol{e}, \boldsymbol{l_F}]). \tag{8}$$

Note that $\boldsymbol{W}_{ze} \in \mathbb{R}^{H\times H}, \mathbf{W}_{zl} \in \mathbb{R}^{N_d+N_f\times H}, \boldsymbol{b} \in \mathbb{R}^H$, and $\boldsymbol{W}_h \in \mathbb{R}^{H+N_d+N_f\times H}$ are the parameters of g, σ is the sigmoid function, $\odot$ denotes the element-wise multiplication, and h is a component-wise nonlinearity. The scoring function $f(\boldsymbol{e}_i, \boldsymbol{e}_j, \boldsymbol{r}_k)$ would be replaced with $f(g_{lf}(\boldsymbol{e}_i, \boldsymbol{l}_{\boldsymbol{F}_i}), g_{lf}(\boldsymbol{e}_j, \boldsymbol{l}_{\boldsymbol{F}_j}), \boldsymbol{r}_k)$. The scoring function f, without loss of generality, could be a scoring function from any latent feature model such as DistMult and ComplEx shown in Equation 1 and Equation 2 respectively.

4.3. Computational Complexity

Note that given a KG, the numerical features of entities, as discussed in Section 4.1, are pre-computed, and hence, the major part of the computational cost of LitKGE comes from training the model depicted in the architecture in Figure 3. The cost can be computed in terms of the number of parameters in the model. LitKGE, similar to LiteraLE, introduces some overhead in the number of parameters as compared to the base model. The overhead is the number of parameters in the function g_{lf} in Equation 6. Specifically, there are $2H^2 + 2H(N_d + N_f) + H$ additional parameters corresponding to the dimensionality of W_h, W_{ze}, W_{zl}, and b in Equation 7 and 8. Hence, given g_{lf} and H, the number of additional parameters of LitKGE grows in $O(N_d + N_f)$, that is, linear to the number of attributes (a.k.a data relations) and features in the KG.

5. Experiments

In this section, the details of experimentation including the datasets, the experimentation settings, and the results are discussed. Our implementation and the datasets are made publicly available through GitHub[2].

5.1. Datasets

Three standard datasets, namely, FB15K-237, YAGO3-10, and LitWD48K have been used to evaluate the performance of the proposed LitKGE LP approach. FB15K [4] is a subset of Freebase [3] describing facts about movies, actors, awards, sports, and sports teams. FB15K-237 is created by removing the inverse relations from FB15K [4]. YAGO3-10 is extracted from the YAGO3 knowledge graph, which mostly consists of triples describing people such as citizenship, gender, and profession. For YAGO3-10, the numerical literals published in YAGO3-10-plus [19] and for FB15K-237 the numerical literals provided by LiteralE [14] are used for the experiments in this work. Differently from FB15K-237 and YAGO3-10, LitWD48K is a recent benchmark dataset extracted from Wikidata by explicitly designing it to contain literals so as to evaluate the performances of LP models. The entities of the dataset LitWD48K contain People, Geography, Entertainment, Transportation, Sport, Travel, Business, and Research. The relations are object properties connecting the entities with each other whereas the attributes are datatype properties connecting entities to numerical literals. The statistics of these datasets are given in Table 1.

5.2. Experiment Setting

Note that the experiments are conducted using the scoring functions from two different base models DistMult and ComplEx and the LitKGE models corresponding to these functions are referred to as DistMult-LitKGE and ComplEx-LitKGE. In order to train the models, the strategy from LiteralE[14] is adopted. The hyperparameters used in our experiments across all datasets are: learning rate 0.001, batch size 128, embedding size 100, embedding dropout probability 0.2, and label smoothing 0.1. DistMult-LitKGE is trained for a maximum of 200 epochs on FB15K-23 and YAG03-10 datasets and 300 epochs on LitWD48K whereas ComplEX-LitKGE is trained for a maximum of 100 epochs on all datasets. More details on the hyper-parameters for the feature generation pipeline are provided in the publicly accessible resources on GitHub.

[2] https://github.com/GenetAsefa/LitKGE

Table 1
The statistics of the datasets used in the experiments in this paper.

	FB15K-237	YAGO3-10	LitWD48K
#Entities	14,541	123,182	47,998
#Relations	237	37	257
#Attributes	121	5	297
#Relational Triples	310,116	1,089,040	336,745
#Numerical Attrib. Triples	70,257	111,406	324,418
#Train	272,115	1,079,040	303,117
#Test	17,535	5,000	16,838
#Valid	20,466	5,000	16,838

5.3. Training

The same training procedure used in LiteralE which is known as 1-N scoring approach is adopted to train the LitKGE models. For every triple (e_i, e_j, e_k) in the KG, the score for $(e_i, e_j^{'}, e_k)$, $\forall e^{'} \in E$ is computed by applying the extended scoring function f. Then, the sigmoid function is applied to the resulting score (i.e., $p = \sigma \circ f$), in order to interpret the result as the probability of the existence of a given triple. Finally, the probability vector $P \in [0, 1]^{N_e}$ collects the probabilities computed w.r.t. all $e^{'} \in E$. Finally, the training is performed by minimizing the binary cross-entropy loss between p and the ground truth labels $y \in 0, 1^{N_e}$ which indicates the existence of triples $(e_i, e_j^{'}, e_k)$, $\forall e^{'} \in E$. Given p_x and y_x as the predicted probability and the given truth value for the x-th element in the set $\{(e_i, e_j^{'}, e_k), e^{'} \in E\}$ respectively, the loss function is defined as shown in Equation 9 and optimized using the Adam [13] optimizer.

$$L(p, y) = -\frac{1}{N_e} \sum_{x=1}^{N_e} (y_x \log(p_x) + (1 - y_x) \log(1 - p_x)) \tag{9}$$

5.4. Evaluation

In order to evaluate the performance of the proposed approach on the LP task, the standard setup in other similar studies including LiteralE is followed. For each triple (e_i, e_j, r_k) appearing in the test set, a set of corrupted/negative triples by either replacing the head entity e_i or the tail entity e_j with any other entity $e^{'} \in \mathcal{E}$. The scores are computed for the corrupted triples as well as the true triple. Then, all triples with respect to their scores are ranked and evaluated using the standard evaluation metrics: Mean Reciprocal Rank (MRR), Hits@1, Hits@3, and Hits@10.

5.5. Results and Discussion

The statistics of the generated features for the datasets FB15K-237, YAG03-10, and LitWD48K with the proposed pipeline in this work, are presented in Table 2. Moreover, the results of the LP experiments conducted on these datasets are provided in Table 3. Overall, as these results indicate, the model LitKGE outperforms all the other models across all datasets with respect to all metrics using the DistMult scoring function. The results are discussed in detail as follows:

LitKGE vs. Base models (DistMult and ComplEx): DistMult-LitKGE outperforms DistMult on all three datasets with respect to all metrics. Specifically, applying LitKGE on top of DistMult improves the MRR score by 1.18%, 11.87%, and 10.38% on LitWD48K, FB15K-237, and YAGO3-10 datasets, respectively. Similarly, applying LitKGE on top of ComplEx improves the MRR score with 1.87% and 3.97%, respectively on LitWD48K and FB15K-237. It can be noted that applying ComplEx-LitKGE does not outperform ComplEx on YAGO3-10. This might be attributed to the fact that the ComplEx base model already achieves higher performance than DistMult. Note that this is also the case with ComplEx-LiteralE, i.e., ComplEx-LiteralE does not improve ComplEx on YAGO3-10.

Table 2

Analysis of the features generated for the entities in the three datasets. #feat denotes the number of unique features and #feat-entries is the number of entries with these features for the entities in the corresponding dataset. #feat-max, #feat-min, and #feat-median represent the maximum, minimum, and median of the occurrences of the features.

	FB15K-237	YAG03-10	LitWD48K
#feat	11	5	828
#feat-entries	2,337	10,151	192,035
#feat-max	1,127	5,072	15,289
#feat-min	8	347	2
#feat-median	170	1,403	12

Table 3

LP results on FB15K-237, YAGO3-10, and LitWD48K. The best values are highlighted in bold text.

	FB15K-237				YAGO3-10				LitWD48K			
	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
DistMult	0.282	0.203	0.309	0.438	0.466	0.377	0.514	0.653	0.336	0.264	0.360	0.480
ComplEx	0.290	0.212	0.317	0.445	0.493	0.411	0.536	0.649	0.315	0.248	0.341	0.442
KBLN	0.301	0.215	0.333	0.468	0.487	0.405	0.531	0.642	-	-	-	-
MTKGNN	0.285	0.204	0.312	0.445	0.481	0.398	0.527	0.634	-	-	-	-
DistMult-LiteralE	0.317	0.232	0.348	0.483	0.479	0.4	0.525	0.627	0.331	0.258	0.352	0.480
ComplEx-LiteralE	0.305	0.222	0.336	0.466	0.485	0.412	0.527	0.618	0.317	0.238	0.343	0.475
DistMult-LitKGE	**0.320**	**0.234**	**0.354**	**0.488**	**0.520**	**0.446**	**0.563**	**0.653**	**0.340**	**0.266**	**0.363**	**0.491**
ComplEx-LitKGE	0.302	0.219	0.333	0.465	0.481	0.406	0.522	0.615	0.321	0.243	0.344	0.480

LitKGE vs. LiteralE: When comparing LitKGE with LiteralE, it is seen that LitKGE outperforms LiteralE across all three datasets with the DistMult scoring function with respect to all metrics. DistMult-LitKGE improves DistMult-LiteralE's MRR score by 2.56%, 0.94%, and 7.88% on LitWD48K, FB15K-237, and YAGO3-10 datasets, respectively. Similarly, applying LitKGE on top of ComplEx improves the MRR score by 1.25% on LitWD48K dataset. As it is mentioned above in the context of LitKGE vs. base models, the reason for ComplEx outperforming ComplEx-LitKGE on YAGO3-10 might be attributed to the already higher performance of the ComplEx base model over DistMult. The same reasoning can be applied to clarify why ComplEx-LitKGE does not achieve better results than ComplEx-LiteralE on FB15K-237 and YAGO3-10 datasets; instead, it delivers comparable performance. Another factor contributing to this might be the utilization of hyperparameter tuning, which was conducted in [6] and also adopted in this work and LiteralE [14] for carrying out the experiments.

Impact of the feature matrix: As shown in Table 2, the LitWD48K dataset has many features (i.e., 828) as compared to the other datasets, and also it has as large as 192,035 entries which makes the resulting feature matrix less sparse. This contributes to the fact that LitKGE outperforms all the SOTA models on this dataset. Note that, unlike LitWD48K, FB15K-237 and YAGO3-10 are not created specifically for the evaluation of KGE models with literals, i.e., not all of their entities have literals. However, these datasets could benefit from the feature generation approach in LitKGE since LitKGE also generates features for those entities which do not have any literals associated with them. This gain is observed in the LP results obtained on these datasets with LitKGE where LitKGE outperforms the SOTA models.

Impact of filtering while generating features: As discussed in Section 4.1, those features which have values for only one entity are filtered out. This is performed in order to make the feature matrix less sparse. An ablation study is conducted on the LitWD48K dataset with the DistMult-LitKGE model in order to show the impact of performing the filtering step in the feature generation pipeline in Figure 2. As the results in Table 4 indicate, DistMult-LitKGE outperforms $DistMult-LitKGE_{unfiltered}$ with respect to all metrics, mainly with hits@10 metrics which is improved by 3.46%. This improvement is attributed to the filtering step in the pipeline during which 269 property_paths/features occurring only once are removed. This proves that including filtering in the pipeline plays its role in reducing sparsity.

Table 4

Comparison of the LP results on LitWD48K dataset with and without applying the filtering step in the feature generation pipeline. DistMult-LitKGE is with filtering whereas DistMult-LitKGE$_{unfiltered}$ is when filtering is not used.

	MRR	Hits@1	Hits@3	Hits@10
DistMult-LitKGE	**0.340**	**0.266**	**0.363**	**0.491**
DistMult-LitKGE$_{unfiltered}$	0.334	0.263	0.357	0.474

Experiment with a different scoring function In this work, as shown above, DistMult and ComplEx are chosen as scoring functions for LitKGE based on the following considerations. Firstly, DistMult and ComplEx are widely adopted and have been demonstrated to be effective latent feature methods, as reported in [14]. By utilizing these scoring functions, it is possible to compare the results obtained with LitKGE against the SoTA models such as LiteralE-DistMult and LiteralE-ComplEX. Secondly, it is convenient to analyze the results obtained using datasets with different characteristics, such as comparing the symmetric and inverse handling capabilities of these models on various datasets. Thirdly, as discussed in [9] where a comparison of datasets for KGE with literals is provided, FB15K-237 is better than YAGO3-10 and FB15K to evaluate KGE models with literals. Besides, according to the results presented in [14], the DistMult and ComplEx scoring functions perform better than ConvE on the FB15K-237 dataset. Therefore, DistMult and ComplEx are chosen in this work to evaluate LitKGE.

Table 5

Comparison of LitKGE with ConvE scoring function against the SOTA models on the Yago3-10 dataset

	MRR	Hits@1	Hits@3	Hits@10
ConvE-LiteralE	0.525	0.448	0.572	0.659
ConvE-LitKGE	**0.540**	**0.467**	**0.583**	**0.671**

However, it is very important to note that the methods DistMult and ComplEx are a representative selection and LitKGE can also be adapted to alternative latent feature methods, such as ConvE. In order to support this argument, an additional experiment is conducted with the ConvE scoring function on YAGO3-10 and the results are presented in Table 5. Note that YAGO3-10 dataset is selected due to the fact that ConvE performs best only on this dataset as compared to the other datasets with the current approaches [14]. The results presented in Table 5 indicate that LitKGE with the ConvE scoring function outperforms the current best-performing model ConvE-LiteralE proposed in [14].

Limitations The primary constraint of this study lies in its reliance on a Knowledge Graph (KG) that contains some numerical literals associated with its entities. In cases where the KG lacks such numerical literals, the feature generation method proposed in this work is not applicable for generating numerical features for entities.

6. Conclusion and Future Work

This study presents a novel approach named LitKGE which is designed to improve the way KGE models utilize literals to perform LP on KGs. Its main focus is on making implicit information explicit so that KGE models could leverage it. LitKGE is tested on LitWD48K, FB15K-237, and YAGO3-10 datasets and it outperforms all SOTA models across all these datasets. The following are possible future works in this direction:

– Exploring schema definitions and constraints of properties to deal with those KGs with possible duplicate paths and to handle sparsity in the feature matrix.
– Extending LitKGE by fusing the relational triples and the numeric literals with short textual literals such as labels and aliases and also long textual literals such as the description of entities.
– Studying explainability of the LitKGE model.
– Exploring the advantages of applying the proposed feature generation approach for the construction of knowledge graphs.

References

[1] F. Bakhshandegan Moghaddam, C. Draschner, J. Lehmann and H. Jabeen, Literal2feature: An automatic scalable rdf graph feature extractor, in: *Further with Knowledge Graphs*, IOS Press, 2021, pp. 74–88.
[2] M. Blum, B. Ell and P. Cimiano, Exploring the impact of literal transformations within Knowledge Graphs for Link Prediction, in: *The 11th International Joint Conference on Knowledge Graphs (IJCKG)*, 2022.
[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge and J. Taylor, Freebase: a collaboratively created graph database for structuring human knowledge, in: *Proceedings of the ACM SIGMOD international conference on Management of data*, 2008.
[4] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston and O. Yakhnenko, Translating Embeddings for Modeling Multi-Relational Data, in: *NIPS*, 2013.
[5] C. Chung, J. Lee and J.J. Whang, Representation Learning on Hyper-Relational and Numeric Knowledge Graphs with Transformers, *arXiv preprint arXiv:2305.18256* (2023).
[6] T. Dettmers, P. Minervini, P. Stenetorp and S. Riedel, Convolutional 2d knowledge graph embeddings, in: *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[7] A. García-Durán and M. Niepert, KBlrn: End-to-End Learning of Knowledge Base Representations with Latent, Relational, and Numerical Features, in: *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, A. Globerson and R. Silva, eds, AUAI Press, 2018, pp. 372–381.
[8] G.A. Gesese, Leveraging literals for knowledge graph embeddings, PhD thesis, Karlsruher Institut für Technologie (KIT), 2023. doi:10.5445/IR/1000161136.
[9] G.A. Gesese, M. Alam and H. Sack, LiterallyWikidata - A Benchmark for Knowledge Graph Completion Using Literals, in: *The Semantic Web – ISWC 2021*, Springer International Publishing, Cham, 2021, pp. 511–527.
[10] G.A. Gesese, H. Sack and M. Alam, RAILD: Towards Leveraging Relation Features for Inductive Link Prediction In Knowledge Graphs, in: *The 11th International Joint Conference on Knowledge Graphs (IJCKG)*, 2022.
[11] G.A. Gesese, R. Biswas, M. Alam and H. Sack, A Survey on Knowledge Graph Embeddings with Literals: Which model links better Literal-ly?, *arXiv preprint arXiv:1910.12507* (2019).
[12] A. Grover and J. Leskovec, Node2vec: Scalable Feature Learning for Networks, in *KDD '16*, Association for Computing Machinery, New York, NY, USA, 2016, pp. 855–864–. ISBN 9781450342322. doi:10.1145/2939672.2939754.
[13] D.P. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, in: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds, 2015. http://arxiv.org/abs/1412.6980.
[14] A. Kristiadi, M.A. Khan, D. Lukovnikov, J. Lehmann and A. Fischer, Incorporating literals into knowledge graph embeddings, in: *International Semantic Web Conference*, Springer, 2019, pp. 347–363.
[15] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer et al., DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia, *Semantic Web* **6**(2) (2015), 167–195.
[16] M. Li, N. Gao, C. Tu, J. Peng and M. Li, Incorporating Attributes Semantics into Knowledge Graph Embeddings, in: *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2021, pp. 620–625. doi:10.1109/CSCWD49262.2021.9437876.
[17] F. Mahdisoltani, J. Biega and F.M. Suchanek, YAGO3: A Knowledge Base from Multilingual Wikipedias, in: *CIDR*, 2015.
[18] G. Mann, A. Dsouza, R. Yu and E. Demidova, Spatial Link Prediction with Spatial and Semantic Embeddings, in: *International Semantic Web Conference*, Springer, 2023, pp. 179–196.
[19] P. Pezeshkpour, L. Chen and S. Singh, Embedding Multimodal Relational Data for Knowledge Base Completion, in: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2018, pp. 3208–3218.
[20] P. Preisner and H. Paulheim, Universal preprocessing operators for embedding knowledge graphs with literals, *arXiv preprint arXiv:2309.03023* (2023).
[21] Z. Tan, B. Zhou, Z. Zheng, O. Savkovic, Z. Huang, I.-G. Gonzalez, A. Soylu and E. Kharlamov, Literal-Aware Knowledge Graph Embedding for Welding Quality Monitoring: A Bosch Case, *arXiv preprint arXiv:2308.01105* (2023).
[22] Y. Tay, L.A. Tuan, M.C. Phan and S.C. Hui, Multi-Task Neural Network for Non-Discrete Attribute Prediction in Knowledge Graphs, in: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, Association for Computing Machinery, 2017, pp. 1029–1038–. ISBN 9781450349185.

[23] K. Toutanova and D. Chen, Observed versus latent features for knowledge base and text inference, in: *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 2015.
[24] D. Vrandečić and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Communications of the ACM* **57**(10) (2014), 78–85.
[25] Q. Wang, Z. Mao, B. Wang and L. Guo, Knowledge Graph Embedding: A Survey of Approaches and Applications, *IEEE Transactions on Knowledge and Data Engineering* **29**(12) (2017), 2724–2743.
[26] Y. Wu and Z. Wang, Knowledge Graph Embedding with Numeric Attributes of Entities, in: *Proceedings of The Third Workshop on Representation Learning for NLP*, Association for Computational Linguistics, 2018, pp. 132–136.