Updating Knowledge Graph Embeddings by Intermediate Estimations on Numerical Attributes

Roderick van der Weerdt*, Victor de Boer, Ronald Siebes and Frank van Harmelen

Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands

Abstract

Graph embedding methods are used to create representations of knowledge graph entities in a high-dimensional vector space. These embeddings can be used in downstream tasks such as classification or link prediction. However, typically graph embedding methods require training on the entire knowledge graph, making them less efficient when a knowledge graph is expected to be dynamic, such as with IoT measurements graphs, which are updated throughout the day with new numerical measurements. This paper introduces a method for efficiently creating embeddings for new knowledge graph entities, without retraining the embedding model. The proposed method estimates an embedding for new entities, by averaging the embedding of the k nearest neighbors, where nearness is based on numerical attributes associated with the entities. We investigate the performance of this method, both on a synthetic knowledge graph, and five real-world knowledge graphs. In these experiments, we employ RDF2vec as the embedding method, and classification as the downstream task. We compare three distance measures for determining nearest entities. We observe a trade-off between the accuracy and efficiency of estimating the embeddings and retraining the embedding model. Resulting in a significant decrease of training time, compared to retraining the full model, with only a relatively small reduction in precision. Results show that in cases where the attributes are representative enough our method is an effective and efficient method to incrementally adjust graph embeddings.

Keywords

Knowledge Graphs, Graph Embedding Model, Dynamic Data, IoT measurement data

1. Introduction

Graph embedding methods are used to create representations of knowledge graph entities and their relations in a high dimensional vector space [1]. Those embeddings can be used in downstream tasks such as classifiers or link prediction [2]. To learn entity embeddings, methods such as RDF2Vec [3] learn in a single process over an entire knowledge graph. This means that when new entities are added (for example: when new museum objects are added to a collection [4] or new measurements are added to an IoT measurement graph [5]), all embeddings need to be retrained with the new knowledge graph.

Graph embedding models are typically not designed to handle a growing knowledge graph, and will have to be retrained in order to create embeddings for new information [6]. This also means that the downstream applications that use the embeddings will also have to be retrained in order to correctly interpret the new embedding representations. Retraining everything is a time and energy consuming process [7], which we try to delay by proposing an alternative method of estimating the embeddings for new entities. We do this by reusing the embeddings from similar entities that are already available in the embedding space. Similarity between existing entities and new entities is based on numerical attributes retrieved from the knowledge graph. For example, a museum object has its size dimensions as numerical attributes, a movie has its duration, or for IoT measurements graphs it could be the temperature values measured by various sensors.

We design this method for a usage scenario where new entities are expected to appear in a continuous fashion, such as the aforementioned museum or IoT measurement case, or other data stream situations.

EKAW 2024: EKAW 2024 Workshops, Tutorials, Posters and Demos, 24th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2024), November 26-28, 2024, Amsterdam, The Netherlands.

*Corresponding author.

r.p.vander.weerdt@vu.nl (R. v. d. Weerdt)

0000-0002-1125-1126 (R. v. d. Weerdt); 0000-0001-9079-039X (V. d. Boer); 0000-0001-8772-7904 (R. Siebes); 0000-0002-7913-0048 (F. v. Harmelen)

We detail the method and its parameters in Section 3.

We evaluate this method both on a synthetic knowledge graph and on real-world knowledge graphs (Sections 4 and 5). The synthetic knowledge graph allows us to adjust the amount of noise in the numerical values of the graph. By adjusting this noise level we can explore the effect of noise between the numerical values and the classification targets, on the effectiveness of the embedding estimation method. We further test the embedding estimation method with five different knowledge graphs from different domains. In both experiments, we examine the trade-off between loss of accuracy due to the estimated nature of the new embeddings and time improvements.

2. Background

Knowledge graphs are data structures consisting of entities and relations connected in a graph, with the entities as nodes, and the relations as edges [8]. The data can be expressed as RDF triples, which use the following shape: <*entity1, relation, entity2*>. In this section we present work related to changing graphs (Section 2.1 and 2.2), the related topic of inductive learning (Section 2.3) and different methods that can be used for entity representation learning (Section 2.4 and Section 2.5)).

2.1. Learning on Dynamic Graphs

Learning representations for entities in dynamic knowledge graphs has been researched in the past. Dynamic graph convolutional networks [9] learn from the dynamic aspect of a graph. By considering each timestamp as a snapshot of the graph and learning over all these versions of the graphs the temporal aspect can be captured. These networks are able to learn from dynamic data, but still require all data to be available when they are being trained. Instead our method allows for new information to be processed directly. Note that in contrast to [9], our method is limited to dealing with incremental addition of entities only, and does not deal with other types of alterations, such as restructuring or entity removal.

2.2. Learning on Temperal Graphs

By adding timestamps to every property of the graph, a temporal knowledge graph is created, which can also be utilized to represent dynamic knowledge graphs. In the T-GAP model [10], for example, the authors use the timestamp of the query to determine which parts of the graph are most relevant in answering the query. Even though adding such timestamps introduces a temporal element, this is not what we want to capture in this research. Such models for temporal knowledge with timestamps graphs do not learn from new knowledge.

There are frameworks created to handle data streams for situations where data is expected to be added regularly, such as Kafka-ML [11]. This framework provides a way to combine data streams with machine learning (ML) methods. The data from a data stream is stored in the framework, allowing ML methods to use the data for training. Contrary to our method, in this framework, new data received after training the method will not be taken into account for the training of the model. Our proposed estimation method can therefore be seen as an addition to the framework.

2.3. Inductive Learning

The field of inductive learning focuses on creating methods to use with previously unseen entities or relations. For example, by creating an embedding generator for entities in a graph, and then using it to create embeddings for new entities in that graph [12].

Shi et al. [13] propose ConMask, a model to create embeddings for unseen entities by exploiting the string names of entities. They first create embeddings for the known entities and then train a second model, which uses the entity names (as embeddings retrieved from a language model) as input, and

the created embeddings as output. When presented with unseen entities ConMask is used to create embeddings for them, which can then be used in combination with the original entities.

Another inductive learning approach is proposed by Gesese et al.: RAILD [14], which in addition to learning representations to entities, also learns representations for relations, for the specific task of link prediction. The textual representations of entities and relations are encoded using a language model.

The work of Galkin et al. [15] takes inductive learning a step further by introducing ULTRA, which not only works on unseen entities, but on completely unseen graphs. It uses a pre-trained foundation model that creates representations for relations in a graph based on the interactions between those relations. Since it is pre-trained on another graph it does not have be trained on new entities.

What sets our approach apart from previous inductive learning approaches is that we leverage the numerical properties of the graph and the entities in order to make an estimation for the entity embedding. Additionally, we do not use a learning approach for the estimation part, using only information already available, while inductive learning trains requires more preparation, such as an additional model for the creation of embeddings for unseen entities [16].

2.4. GCNs

Graph convolutional neural networks (GCNs) are used in approaches such as GraphSAGE [16] and the more recent Dynamic Knowledge Graph Embedding (DKGE) [17] and Lifelong Knowledge Graph Embedding (LKGE) [18], to learn representations for entities. By only taking into account the direct context for each entity they learn embeddings for entities without considering the entire graph. Therefore, when a new entity is added to the graph, an embedding can be created for it by only considering its new context, without altering the model or the other embeddings. However, GCNs need to be trained end-to-end, while our method considers graph embedding methods that are trained independently of various possible downstream tasks. Additionally the embedding estimation method is able to capture longer range dependencies between entities, by using the numerical values of the entities, which is a challenge in GCN-based approaches.

Leveraging modalities was investigated in multimodal learning [19], where the authors create a pipeline with a dedicated neural network for specific modalities, and then combining their embeddings with a GCN to learn more specific representations. However these methods were trained end-to-end, with the learned representation only being relevant for a specific task. Our estimation method is able to use embeddings created to represent entities from the knowledge graph independently of a task. Additionally our method would not require any changes to the pipeline in place, only the addition of the method.

2.5. RDF2Vec

RDF2Vec [1] is a graph embedding model that generalises the Word2Vec [20] algorithm to work with graphs. It learns representations from knowledge graph entities by considering the entities as “words”, and random walks through the graph as “sentences”. Entities are embedded in a vector space based on the nodes that occur in their neighbourhood in such random walks. During training, more similar entities will be embedded closer together in the embedding space, and dissimilar entities will be further apart. Because all embeddings affect each other during training it is not possible to add new embeddings without retraining all the other embeddings. We primarily design our method to extend RDF2Vec with this capability.

Recent work by [21] updates both existing and new embeddings by what the authors refer to as *resuming* the training of a RDF2Vec model, by which they mean re-starting the training process: retrieving new walks for all new entities (or relations), and retraining the existing RDF2Vec model with these new walks. While this approach shows promising results, even out performing the retrained model of evaluation tasks, it differs to the embeddings estimation method proposed in this paper in terms of the kind of scenario where both would be applied. Where the approach from Hahn et al. suits

large knowledge graph which have a large update, our embedding approach suits the knowledge graphs with a small update, where the estimated embedding is used as an interim solution.

Our estimation method intends to supplement RDF2Vec, or any other task agnostic graph embedding model. We base the estimated embeddings for new entities on the existing embeddings, by leveraging the related numerical attributes of the entities.

3. The Embedding Estimation Method

In this section, we present the main contribution of this paper, which is the embedding estimation method. The setup of the experiment and its results follow in Section 4 and 5, respectively. The method estimates an embedding for new entities based on the embeddings of most similar previously embedded entities (we refer to the latter as "original entities"). Similarity between original entities and new entities is determined with a distance measure (Section 3.2) between the numerical attribute values of the entities.

3.1. Estimation Method

Algorithm 1 provides pseudocode of our method for one individual new entity. Besides the new entity it requires: all original entities (including their previously calculated embeddings) and parameter k, which determines the number of original entities used to calculate the embedding for the new entity.

The first step is calculating the distance between the attribute values of the new entities and each individual original entity. By sorting these distances in ascending order we create a ranking of original entities most similar to every new entity. The embeddings of the k most similar entities are averaged to create the embedding for the new entity. In other words, the distance measure between the attributes is used as a *proxy* for the (ideal) distance in the embedding space.

In this implementation of the estimation method we do not apply normalization. The estimation method assumes that the attribute values will be a reasonable representation for the entities, using them to estimate a new embedding for each new entity.

Algorithm 1 Calculates the estimated embedding for a new entity, based on the similarity between its attributes and those of the original entities.

$entity_{new}$ = 1 new entity for which we want an embedding
$entities_{original}$ = All original entities with embeddings
k = number of nearest entities to consider

function EMBEDDING_ESTIMATOR($entity_{new}, entities_{original}, k$)
  for all $entity_{original} \in entities_{original}$ **do**
    $dist \leftarrow$ DISTANCE_MEASURE($entity_{original}.attributes, entity_{new}.attributes$)
  end for
  $matches$ = SORT_ASCENDING($dist$)
  $k\_matches$ = GET_TOP_K($matches, k$)
  $embs$ = GET_EMBEDDINGS($k\_matches, entities_{original}$)
  $embedding_{new}$ = AVERAGE($embs$)
  return $embedding_{new}$
end function

3.2. Distance Measures

We implement three different distance measures: Euclidean, Chebyshev and Manhattan distance. Below we give a short description of each distance measure. We treat A and B as vectors. The length of the vectors depends on the number of numerical values the knowledge graph contains.

Euclidean distance = $\sqrt{\sum (A - B)^2}$, The shortest line between two points.

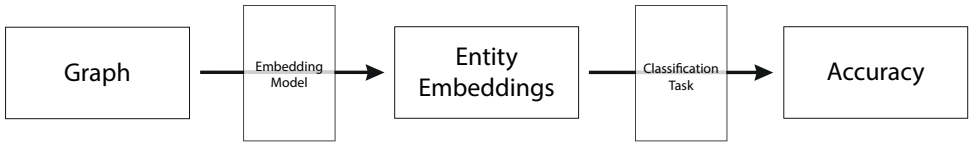

Figure 1: Example scheme of a typical pipeline used in the situation where the embedding estimation method can be used. An embedding model learns embeddings for entities in a graph, which are used in a classification task.

Chebyshev distance $= \max |A - B|$, The longest distance between two points, in any one of the dimensions of the points.

Manhattan distance $= \sum |A - B|$, the summed distance between two points over all dimensions of the points.

4. Experiment

In order to test the effectiveness of the estimation method, we measure the accuracy and runtime of the classification task on 10 variants of a synthetic knowledge graph and five real-world knowledge graphs, comparing the estimation method against the retraining method. The setup of this experiment is described in Section 4.1.

The creation of the ten variations of the synthetic knowledge graph are detailed in Section 4.2.1. By adjusting the noise level in the synthetic knowledge graph we will determine the effect of noise on the effectiveness of the estimation method, and the effect of changing value k.

The five domain knowledge graphs presented in Sections 4.2.2, 4.2.3 and 4.2.4 demonstrate how the embedding estimation method performs on real-world knowledge graph.

4.1. Setup

The setup of the experiment assumes there already exists an *original* pipeline, from which multiple elements will be reused by the estimation method. An example of how we expect such a pipeline to look like is given in Figure 1.

For the embedding estimation method we reuse the original embeddings and entities from the graph to create the embedding estimator, which uses the new entities to create the new embeddings. By reusing the classification method, new accuracies are calculated. We use 21 values for parameter k ranging between 1 and 250, and n, with n being the total amount of original entities. This results in the average embedding of all original entities to be used for all new entities. This is added as a baseline. The experiments are repeated, with all three distance measures, to determine the effect of choosing the distance metric.

The *retraining method* combines the new and original knowledge graph and original entities to completely retrain all models created in the original pipeline, reusing nothing.

RDF2Vec [1] is used to compute the embedding model, using the PyRDF2Vec library [3], with a walk-length of 2 and 25 walks per entity. The model is trained for 20 epochs. Each specific knowledge graph has a specific downstream classification task, described in the following sections. In each case, a MLP is used to create the classification model, by deploying the pyTorch library [22]. The model consists of two hidden ReLu layers, each with 512 nodes, trained for 40 epochs. All code is available on GitHub[1].

The experiment is performed with: a synthetic knowledge graph using the SAREF ontology, two smart building knowledge graphs and a selection of three knowledge graphs from different domains, based on the availability of attributes, from KGBench [23]. Numerical attributes of entities are collected from the knowledge graph, both for the original entities and new entities, by querying the knowledge graphs.

[1] https://github.com/RoderickvanderWeerdt/Embedding_Estimator

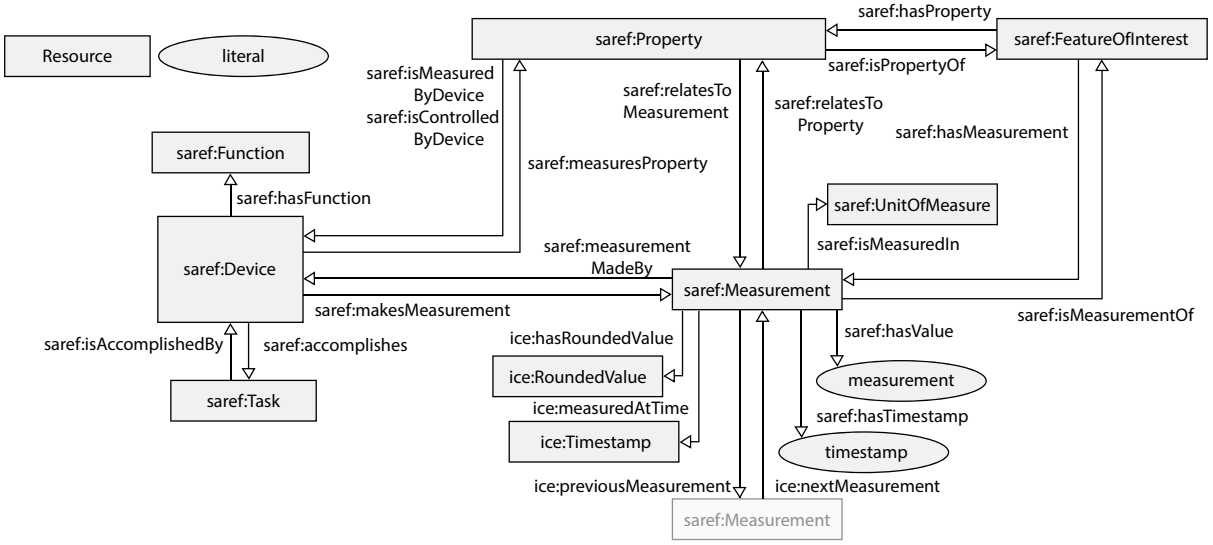

Figure 2: Overview of the synthetic knowledge graph template, using the SAREF ontology [24] (with prefix saref) the extension for learning [5] (with prefix ice). The previous measurement resource is grayed out, to indicate this is another measurement, of which we do not show all properties.

4.2. Datasets

For the experiments we use six different knowledge graphs. First we discuss the creation of the synthetic knowledge graph. Because real world knowledge graph are noisy and incomplete we also perform the same experiments with real world knowledge graphs from multiple domains.

4.2.1. Synthetic Knowledge Graph

In order to test the effect of noise on the embedding estimation method we create a synthetic knowledge graph with adjustable noise levels. The noise level is a value that describes how much noise (random variation) is added to the numerical data in the knowledge graph. Figure 2 shows the template used for the knowledge graph. The synthetic knowledge graph uses the SAREF ontology [24] to model devices with measurements that directly correspond to the target class that we will be predicting. By adding noise (corresponding to the noise level), we gradually decrease this direct relation. The noise level is added to the measurements directly by adding random values between -noise_level and noise_level to each measurement, thereby creating increasing levels of noise to the data. We create 10 synthetic knowledge graphs, with noise levels ranging from 1 through 10. In addition to the SAREF ontology we use the extension proposed in [5] (using the `ice` prefix) to enhance the learnability of the graph.

The synthetic knowledge graph consists of 10 devices making a total of 8000 measurements, resulting in a knowledge graph of 1.2 million triples. The files that are used to create the synthetic knowledge graph are made available online[2].

By making the measurements (and their values) and classes directly related we expect the classification to be very accurate. By increasing the noise level this relation between the values and the classes will decrease, but the relation between the measurements and classes will remain, because the structure from the graph will not be changed. We perform the experiments on 10 variants of the synthetic graph, with noise levels increasing from 1 up to 10.

4.2.2. OPSD Household data

The OPSD household dataset [25] contains the hourly energy consumption of devices in multiple residences. The dataset has been mapped to a knowledge graph [26]. We used the data from one

[2]https://github.com/RoderickvanderWeerdt/Synthetic_Noisy_IoT_KG_creator

residential household (household 4 in the dataset), since it is large enough to be realistic, but not so large that it takes multiple hours to train the embedding model. The attributes in the graph are: general energy export, general energy import, solar panels, dishwasher, electrical vehicle, freezer, washing machine. Each entity in the graph is a timepoint at which energy consumption measurements were taken. The graph contains 0.9 million triples.

Given the continuous source of data from hourly energy consumption updates the knowledge graph can be considered a good representation for the kind of knowledge graph for which our estimation method is intended. The classification task classifies each time point whether the outside temperature was in the warm half of the measurements, or in the cold half of the measurements. Although this classification task is not a hard task, we consider it a useful task to show the capabilities of the embedding estimation method.

4.2.3. OfficeGraph

Similarly to OPSD, OfficeGraph contains hourly smart building measurements from devices [27]. But instead of only energy measurements, OfficeGraph contains nine heterogeneous types of measurements made by IoT devices, such as: temperature, CO_2 levels and humidity measurements. Due to memory constrains we only use the measurements from devices on the 7th floor, which allows experiments to be performed on a consumer grade machine, instead of having to resort to a super computer or distributed computing environment. The resulting graph contains 13 devices and consists of 3.9 million triples.

Using the local outside temperature we create a similar classification task as with OPSD, predicting whether the outside weather is in the warm or cold half of the total measurements.

4.2.4. KG-Bench

We use three different knowledge graphs from the KGBench knowledge graph benchmark repository [23], which includes benchmark tasks and train/test splits: the Amsterdam Museum Dataset (AM), the Dutch Monument Graph (DMG), and the Movie Dataset (MD). These knowledge graphs were chosen because they include numerical attributes in their knowledge graphs, as required by our estimation method. All image hashes and geodata were removed from the graph, since these would not be used in the experiments and it greatly decreases the file size.

The majority of entities lack one or more attributes, e.g. not all movies in MD have a box office value. more details can be found in the KGBench paper [23]. Entities without any attributes were removed since our method provides no way to estimate an embedding for these entities.

DMG. Entities are monuments in the Netherlands, and the knowledge graph contains three attributes for each monument: “year”, “population” (of the city were the monument is located), “codeNational-Monument”. The graph contains 0.7 million triples and the classification task classifies the type of monument, out of five monument classes.

MD. The entities are movies, and knowledge graph contains three attributes: “duration”, “boxoffice”, “cost”. The graph contains 2.4 million triples and the classification task classifies the genre of the movie, out of 12 genres.

AM. The entities are museum objects and the graph contains one numerical value: “dimension”, but it refers to multiple different dimensions which for these experiments are all considered as different attributes. The different dimensions are separated based on the units of measurement used for the dimension: “gram”, “mm”, “cm”, “ml”, “gr”, “kg”, “G”, “gr.”, “m”, “liter”. When multiple similar units are available for one entity they are all stored, and sorted on size, in order to maintain as many attributes as possible. Despite the fact that “gram” and “gr.” refer to the kind of measurement, they are not considered to be identical, as some entities have dimensions with both units. This resulted in 57 attributes. The graph contains 1.1 million triples and the classification task classifies the museum object out of eight museum object classes.

For all the used KGBench knowledge graphs we use the default subsets provided, being the training set and validation set. For DMG and MG, the training set (as defined in the benchmark) is used to

create the original entities, and the validation set is used to create the new entities. For AM the original entities are created similarly, but for the new entities a subset is taken from the validation set, because the complete validation set is larger then the train set. The estimation method is intended to be used when new entities become available, but when many new entities become available the model should be retrained, instead of continuing to accumulate estimated embeddings. When this should happen should be determined per task. For our experiments we use a rule of thumb that the number of new entities should not be more than 20% of the original entities. For AM* we randomly select entities from the validation set, until their are as many entities as 20% from the training set.

4.3. Metrics

In the experiment we use two metrics to determine the impact of the embedding estimation method. The first is the accuracy of the classification task of each knowledge graph, from classifying the new entities. The second is the time in seconds that the method requires to finish calculating the new embeddings. For the embedding estimator this is the time the embedding estimation takes plus the time the classifier takes. For the re-training part of the experiment this is the time it takes to retrain the embedding, retrain the classifier and perform the classification.

5. Results

We present the results of our experiments in three tables and describe them in the following subsections. We first report on the experiments on the synthetic knowledge graph in sections 5.1,5.2 and 5.3 where we explore the effect of k, the distance measure and time gained respectively. Then in Section 5.4, we describe the same results for the five real-world domain knowledge graphs.

5.1. Different sizes of k

In order to test the effect of changing the size of k we perform the experiment with the synthetic knowledge graphs with increasing values for k, ranging from 1 to 250, as can be seen in the first part of Table 1, using the Euclidean distance measure. This is repeated for noise levels from 1 through 10 (as described in Section 4.2.1). We observe two main effects, 1) higher noise levels produce lower accuracies, and 2) synthetic knowledge graphs with higher noise levels achieve their highest accuracies at higher values for k.

By examining the highest accuracy for each noise level we see the accuracy decrease as a result of increasing the noise level. This is expected behavior, because the embedding estimation method relies on the relation between the values in the graph and the target classes. As the noise between these two increases, the accuracy drops.

When increasing the noise level, the highest accuracy is achieved for increasing values of k, showing a correlation between the noise level and optimal k. This correlation is also expected, since if the noise level is higher the k nearest neighbors of the new entity will be less likely to be similar to the new entity and by increasing k this is counterbalanced.

5.2. Results of different distance measures

To explore the impact of changing the distance measure we repeat the same experiments, but instead of using the Euclidean distance measure, we use the Manhattan distance or the Chebyshev distance, which are described in section 3.2). Similarly to the Euclidean distance measure results, when the noise level increases, a higher value for k is needed to reach the highest accuracy. When examining the highest accuracies from the distance measures for each noise level (which are colored green) we see that the Manhattan distance measures achieves the highest accuracies for each noise level.

Table 1
Resulting accuracies of the experiment for changing k, the synthetic noise level and three distance measures. Colors are added for legibility (per distance measure), green cells have the highest accuracy of that column, yellow cells differ at most 0.2% from the green accuracy, and so on, as described in the legend. Row N uses the total number of instances as value for k.

		Noise level									
	k	1	2	3	4	5	6	7	8	9	10
Euclidean distance	1	99.7%	90.4%	75.8%	67.7%	60.7%	59.1%	55.6%	54.3%	53.5%	54.7%
	5	99.8%	93.5%	81.5%	71.5%	64.3%	62.2%	58.2%	58.1%	55.8%	54.6%
	10	99.8%	93.8%	82.5%	73.2%	65.8%	64.2%	60.0%	59.9%	57.9%	55.5%
	15	99.9%	94.1%	82.5%	74.4%	67.5%	64.9%	61.8%	61.3%	58.1%	56.2%
	20	99.8%	94.1%	83.2%	75.1%	69.1%	65.6%	62.9%	61.0%	58.0%	56.3%
	25	99.9%	94.2%	82.8%	75.1%	69.4%	65.8%	64.0%	61.6%	59.2%	56.0%
	30	99.9%	94.2%	83.2%	75.4%	69.4%	66.1%	64.7%	62.6%	59.0%	56.9%
	40	99.8%	93.9%	82.8%	75.8%	69.5%	67.1%	65.0%	62.0%	59.3%	57.1%
	50	99.8%	93.8%	82.8%	75.5%	69.8%	67.5%	64.3%	62.6%	59.5%	57.5%
	60	99.8%	94.0%	82.5%	75.8%	69.6%	67.6%	64.2%	62.3%	59.4%	57.6%
	70	99.8%	93.8%	82.1%	75.9%	69.5%	67.9%	64.3%	62.3%	59.8%	58.0%
	80	99.8%	93.4%	82.0%	76.0%	69.6%	67.8%	63.9%	62.5%	60.0%	58.4%
	90	99.8%	93.5%	81.9%	75.4%	69.9%	67.6%	64.2%	62.6%	59.7%	58.1%
	100	99.8%	93.3%	81.8%	75.7%	70.3%	67.4%	64.3%	62.9%	60.1%	58.3%
	120	99.8%	93.0%	81.6%	75.9%	70.0%	67.5%	64.5%	63.1%	60.1%	58.3%
	140	99.8%	92.7%	81.6%	75.8%	69.3%	67.3%	64.3%	63.5%	60.2%	59.1%
	160	99.8%	92.5%	81.8%	75.7%	69.5%	67.0%	64.8%	63.4%	60.4%	59.0%
	180	99.8%	92.6%	81.6%	75.3%	69.1%	66.8%	64.1%	63.4%	60.4%	59.4%
	200	99.8%	92.5%	81.4%	75.4%	69.5%	67.2%	64.2%	63.6%	60.5%	59.1%
	250	99.8%	92.5%	81.4%	75.4%	69.6%	67.2%	64.3%	63.1%	60.3%	59.5%
	N	49.9%	50.9%	49.7%	49.9%	49.9%	49.4%	49.9%	49.9%	49.4%	49.8%
Chebyshev distance	1	99.6%	87.5%	71.4%	62.3%	58.7%	55.3%	54.9%	53.7%	52.7%	53.3%
	5	99.8%	90.3%	76.4%	66.6%	60.9%	58.9%	57.0%	56.3%	53.8%	54.2%
	10	99.7%	90.8%	77.6%	68.1%	62.8%	61.0%	58.0%	57.6%	55.8%	53.9%
	15	99.7%	90.8%	78.3%	69.8%	62.6%	62.1%	58.6%	58.4%	56.1%	53.8%
	20	99.7%	90.8%	78.9%	69.9%	64.0%	62.2%	60.2%	59.1%	56.7%	53.9%
	25	99.7%	90.6%	78.9%	70.3%	64.4%	62.1%	60.3%	59.0%	56.5%	54.4%
	30	99.7%	90.3%	79.1%	70.5%	64.9%	62.5%	61.1%	59.4%	58.2%	54.2%
	40	99.6%	90.0%	79.1%	70.5%	65.7%	63.3%	61.9%	60.1%	58.4%	55.4%
	50	99.7%	89.6%	79.2%	70.7%	64.9%	63.3%	61.9%	60.5%	59.0%	55.2%
	60	99.7%	89.6%	79.2%	71.4%	64.9%	63.7%	61.7%	60.8%	59.1%	54.9%
	70	99.6%	89.2%	78.9%	71.3%	65.2%	63.1%	61.9%	60.5%	59.1%	55.5%
	80	99.6%	89.2%	78.9%	71.0%	65.4%	63.2%	62.0%	60.5%	59.4%	55.9%
	90	99.6%	89.3%	78.9%	70.9%	65.3%	62.5%	62.2%	60.7%	59.8%	56.0%
	100	99.6%	89.1%	78.3%	70.6%	65.1%	62.7%	61.5%	60.7%	59.6%	56.2%
	120	99.6%	88.7%	78.2%	70.8%	64.5%	62.5%	61.8%	60.9%	60.3%	56.0%
	140	99.6%	88.6%	78.4%	71.1%	64.9%	62.6%	61.5%	61.2%	59.2%	56.6%
	160	99.6%	89.0%	78.2%	70.9%	65.0%	62.7%	61.5%	61.5%	59.9%	57.0%
	180	99.6%	88.7%	78.1%	70.8%	64.6%	63.0%	61.9%	61.5%	59.4%	56.6%
	200	99.6%	88.8%	78.3%	70.4%	64.9%	62.7%	61.8%	61.2%	59.8%	56.6%
	250	99.6%	88.3%	78.4%	70.4%	64.4%	62.7%	61.9%	61.1%	59.2%	56.7%
	N	50.0%	48.3%	49.0%	49.5%	49.7%	49.4%	50.3%	51.6%	50.6%	50.0%
Manhattan distance	1	99.6%	90.1%	79.2%	70.4%	61.8%	60.8%	57.6%	56.9%	55.0%	55.7%
	5	99.8%	93.6%	83.5%	76.2%	66.4%	65.6%	62.2%	60.0%	57.0%	57.4%
	10	99.9%	94.9%	84.1%	77.3%	69.8%	65.8%	63.5%	62.4%	58.6%	56.5%
	15	99.9%	95.0%	85.5%	78.7%	71.8%	67.5%	64.7%	62.6%	60.5%	57.5%
	20	99.9%	95.3%	85.2%	78.8%	71.3%	68.2%	65.9%	63.6%	62.0%	57.9%
	25	99.9%	95.5%	85.6%	79.4%	72.0%	69.3%	66.4%	63.9%	61.0%	57.7%
	30	99.8%	95.3%	85.5%	79.6%	72.9%	69.1%	67.0%	64.1%	61.4%	59.1%
	40	99.8%	95.0%	86.0%	80.1%	73.1%	69.4%	68.0%	63.5%	61.8%	59.8%
	50	99.8%	95.1%	86.0%	80.0%	73.9%	70.0%	68.5%	63.7%	61.5%	59.9%
	60	99.8%	95.0%	85.8%	80.1%	73.8%	70.3%	68.4%	64.2%	61.4%	60.5%
	70	99.8%	95.1%	85.4%	79.7%	74.0%	70.8%	68.0%	63.9%	61.4%	60.7%
	80	99.8%	95.0%	85.4%	79.9%	74.0%	70.4%	68.1%	63.8%	61.5%	60.6%
	90	99.8%	95.0%	85.3%	79.9%	73.6%	70.1%	68.1%	63.9%	62.4%	60.4%
	100	99.8%	95.1%	85.3%	79.3%	73.4%	70.4%	67.9%	63.9%	62.6%	60.7%
	120	99.8%	94.9%	85.1%	78.9%	73.2%	70.5%	68.0%	64.3%	62.2%	60.6%
	140	99.8%	94.7%	85.1%	79.0%	73.3%	69.7%	68.3%	64.7%	62.3%	61.8%
	160	99.8%	94.6%	85.1%	78.8%	73.0%	69.6%	67.6%	64.5%	62.2%	61.5%
	180	99.8%	94.6%	85.0%	78.8%	72.6%	69.7%	67.6%	64.7%	62.8%	61.6%
	200	99.9%	94.5%	85.0%	78.6%	72.8%	69.0%	67.9%	64.4%	62.4%	61.7%
	250	99.8%	94.2%	84.3%	78.6%	72.3%	69.4%	67.5%	64.8%	62.4%	62.1%
	N	50.3%	50.2%	49.1%	50.5%	50.1%	50.6%	49.4%	49.9%	48.6%	47.5%

legend
=top
<0.2%
<1.0%
<5.0%
<10%
>10%

5.3. Estimator versus Re-Training

To examine the trade-off in using the embedding estimation method instead of re-training the pipeline we can compare the accuracy with time consumption for both approaches. Time consumption is the total time it takes to get a classification for a new entity. For re-training this is the time it takes to re-train the embedding, re-train the classifier, and perform the classifications. For the embedding estimation method this is the time it takes to perform the estimation and classification. Table 2 shows for each noise level the accuracy and time consumption, when using the embedding estimation method or re-training the entire pipeline. For each noise level of the synthetic knowledge graph when re-training the pipeline the accuracy remains constant, at 100%. The consistency of the accuracy is expected due to the graph structure not changing from the addition of noise. Since the graph embedding method does not consider the values of the literals. The perfect accuracy is explained by the fact that the synthetic knowledge graph was created to perform well on this task.

For the accuracy of the embedding estimator, we observe that as the noise level increases, the accuracy decreases. This indicates that the embedding estimation method is most useful when the "noise level" of a knowledge graph is low.

Re-training the pipeline takes almost 24 minutes (1437 seconds, as seen in Table 2), while using the embedding estimator for a new instance takes less then a second. This shows that the embedding estimation method is less resource intensive, providing users with a clear trade-off between accuracy and time.

5.4. Diverse Domain Knowledge Graphs

Table 2 visualises the results of the experiments with the domain knowledge graphs. For all knowledge graphs time is saved, being at least 1000 times faster, up to 3300 times, when using the estimation method compared to retraining the embedding model.

Table 3 show a clear distinction between the smart building knowledge graphs and the other domain knowledge graphs. OPSD and OfficeGraph reach their highest accuracy with a low k, while DMG, AM* and MD require a high k to reach their highest accuracy. When we compare this with Table 1, where the synthetic knowledge graphs with higher noise levels reached their highest accuracy with larger values for k, we could argue that this indicates that DMG, AM* and MD have a higher noise level, then OPSD and OfficeGraph. which would make sense, since the numerical attributes of OPSD and OfficeGraph are measurement data, directly representing their entities. While DMG, AM* and MD have

Table 2
Comparison between re-training the entire pipeline and using the embedding estimator, based on the accuracy over the test set and the amount of time consumed in re-training the entire pipeline versus the amount of time consumed in using the embedding estimation for one new instance. The shown embedding estimation results are the highest accuracies from Table 1 and 3 for each knowledge graph.

	Re-training		Embedding estimator		Trade-off	
	accuracy	time (in seconds)	accuracy	time (in seconds)	accuracy	time
datasets	(in %)	embedding+classifier	(in %)	estimation+classifier	loss	gain
synth-1	100.0%	1381 + 56	99.9%	0.3 + 0.01	-0.1%	4,635x
synth-2	100.0%	1381 + 56	95.5%	0.3 + 0.01	-4.5%	4,635x
synth-3	100.0%	1381 + 56	86.0%	0.3 + 0.01	-14.0%	4,635x
synth-4	100.0%	1381 + 56	80.1%	0.3 + 0.01	-19.9%	4,635x
synth-5	100.0%	1381 + 56	74.0%	0.3 + 0.01	-26.0%	4,635x
synth-6	100.0%	1381 + 56	70.8%	0.3 + 0.01	-29.2%	4,635x
synth-7	100.0%	1381 + 56	68.5%	0.3 + 0.01	-31.5%	4,635x
synth-8	100.0%	1381 + 56	64.8%	0.3 + 0.01	-35.2%	4,635x
synth-9	100.0%	1381 + 56	62.8%	0.3 + 0.01	-37.2%	4,635x
synth-10	100.0%	1381 + 56	62.1%	0.3 + 0.01	-37.9%	4,635x
OPSD	72.3%	1268 + 92	83.1%	0.4 + 0.01	10.8%	2,502x
OfficeGraph	76.1%	973 + 53	80.6%	0.3 + 0.01	4.5%	3,309x
AM*	70.5%	615 + 98	54.7%	0.7 + 0.01	-15.8%	1,004x
DMG	62.7%	594 + 56	44.0%	0.3 + 0.01	-18.7%	2,096x
MD	64.4%	471 + 39	60.9%	0.2 + 0.01	-3.5%	2,428x

only indirect numerical attributes.

With DMG, the estimation method is not able to create embeddings that outperform the baseline. The accuracy of attributes for DMG are also below the baseline, indicating that the attributes were not at all representative for the entities of DMG. We analysed the knowledge graph and found that for the available numerical attributes: "year", "population" and "codeNationalMonument" only the "codeNationalMonument" is present for every entity, and this is an identification number, so it is not useful for comparisons with other entities. For this graph the embedding estimation method is not effective.

The results for both MD and AM show that there is a decrease in accuracy when comparing the estimation method with retraining. This result is expected, since these are, after all, estimates based on the limited information from the numerical attributes. However, both outperform the baseline of k=**N**, showing that a meaningful representation is still learned for the entities. Combined with the time gain of using the embedding estimation method we consider this a useful situation for the embedding estimation method.

Table 2 shows that with OPSD and OfficeGraph the accuracy of the estimation method outperforms the accuracy of retraining. When we consider Table 3 we see that OPSD and OfficeGraph also differ in that they reach their highest accuracy with a small value for k (either 1 or 5) as opposed to the other knowledge graphs, which require high values for k. We explain this by examining the classifier, which only in the case of OPSD and OfficeGraph was overfitted on the training data. Because the classifier is not retrained we are using the same embeddings in the embedding estimator which are used in training the (overfitted) classifier, therefor resulting in "better" embeddings than the ones that are kept in the training set, during the re-training of the models. As future research we suggest investigating what the effect is of re-training the classifier with the new entities, as compared to only using the test set for the estimation process.

6. Discussion

The results from OPSD, OfficeGraph and DMG show that the estimation method has requirements from the knowledge graph and entities in order to estimate useful embeddings. The attributes related to the entities that are queried from the knowledge graph need to be representative enough to be able to find the similar entities. When the usage scenario fits these requirements, as it does with MD and AM, the estimation method can be useful as an interim solution, to use new data before having to retrain the pipeline. As mentioned in Section 4.2.4, many entities in the knowledge graphs from KGBench are

Table 3
Results of changing k and distance measures, for five domain datasets. The cells are colored according to the same legend as Table 1, relative to the highest accuracy in each column, with **blue** as a new color to mark the situation where the baseline outperforms the highest accuracy.

Dataset		DMG			AM*			MD			OPSD			OfficeGraph		
Dist. M.		Manh	Eucl	Cheb	Manh	Eucl	Cheb	Manh	Eucl	Cheb	Manh	Eucl	Cheb	Manh	Eucl	Cheb
k	1	28.6%	30.5%	28.6%	10.5%	47.6%	44.6%	30.4%	52.4%	52.3%	83.0%	83.1%	80.5%	78.7%	78.0%	78.2%
	5	30.0%	30.2%	30.0%	20.9%	51.0%	49.3%	46.2%	57.7%	57.7%	81.1%	80.4%	78.7%	80.2%	80.0%	79.8%
	10	37.7%	37.9%	38.0%	10.9%	51.2%	49.4%	52.0%	58.5%	58.5%	80.8%	81.1%	78.7%	80.6%	80.4%	79.5%
	15	27.7%	27.8%	27.6%	17.9%	52.3%	51.0%	53.1%	59.2%	59.2%	80.5%	80.5%	78.0%	80.6%	80.1%	79.2%
	20	27.7%	27.8%	27.7%	11.8%	52.6%	50.6%	54.1%	59.6%	59.6%	80.5%	80.0%	77.7%	80.4%	80.1%	78.5%
	25	27.7%	27.1%	27.7%	32.4%	53.2%	51.1%	54.7%	59.7%	59.7%	80.3%	79.5%	78.1%	80.2%	79.4%	77.7%
	30	27.0%	27.0%	27.1%	34.0%	52.9%	51.5%	57.5%	60.3%	60.3%	80.0%	79.5%	78.1%	80.2%	78.7%	77.0%
	40	37.4%	37.1%	37.3%	34.2%	52.9%	51.6%	58.1%	59.5%	59.5%	79.6%	79.5%	78.4%	79.6%	77.9%	74.9%
	50	37.2%	37.0%	36.7%	34.1%	53.6%	52.2%	59.9%	60.7%	60.7%	79.0%	79.0%	78.1%	78.4%	77.2%	73.3%
	60	36.7%	37.1%	36.6%	34.1%	53.5%	52.2%	59.9%	60.5%	60.5%	79.1%	78.6%	78.1%	78.4%	77.0%	72.0%
	70	36.7%	37.0%	36.7%	34.1%	53.5%	52.1%	59.8%	60.4%	60.4%	78.9%	78.3%	77.4%	78.7%	76.4%	70.9%
	80	37.5%	36.7%	37.5%	34.3%	54.3%	52.5%	60.3%	60.3%	60.3%	78.6%	77.9%	77.2%	78.3%	75.4%	69.8%
	90	37.5%	36.9%	37.5%	34.3%	53.7%	52.5%	60.7%	60.6%	60.6%	78.4%	77.9%	77.0%	78.0%	75.1%	68.7%
	100	37.5%	36.9%	37.5%	34.2%	53.8%	52.4%	60.6%	60.9%	60.9%	78.3%	77.5%	76.9%	77.6%	74.4%	68.2%
	120	38.3%	37.2%	37.6%	34.3%	53.7%	52.8%	60.8%	60.5%	60.5%	78.4%	77.1%	76.0%	76.9%	72.8%	66.9%
	140	39.6%	37.0%	37.6%	34.3%	53.9%	53.3%	60.8%	60.6%	60.6%	78.3%	76.7%	75.7%	76.8%	71.3%	66.2%
	160	41.2%	37.9%	41.2%	34.3%	53.9%	53.4%	60.8%	60.3%	60.3%	78.4%	76.6%	75.5%	76.1%	70.2%	64.9%
	180	42.0%	38.5%	41.3%	34.3%	54.2%	53.1%	60.8%	60.4%	60.4%	78.1%	76.4%	75.0%	75.7%	69.2%	63.7%
	200	42.0%	39.3%	42.0%	34.3%	54.4%	53.3%	60.7%	60.5%	60.5%	77.8%	76.4%	75.0%	75.9%	68.4%	63.5%
	250	44.0%	40.6%	42.0%	34.3%	54.7%	53.3%	60.8%	60.4%	60.4%	77.8%	76.1%	74.7%	75.0%	67.3%	63.5%
	top	44.0%	40.6%	42.0%	34.3%	54.7%	53.4%	60.8%	60.9%	60.9%	83.0%	83.1%	80.5%	80.6%	80.4%	79.8%
k	N	47.8%	47.8%	47.8%	33.9%	39.8%	39.1%	60.8%	60.8%	60.8%	50.3%	50.7%	49.7%	49.7%	49.0%	49.5%

missing one or more numerical attributes. Additionally, we use all available numerical attributes for the entities in each knowledge graph. This results in the inclusion of possibly less desirable numerical attributes, for example with "codeNationalMonument" from DMG. Future research should delve into the selection process of numerical attributes.

For the experiments we compare the resulting accuracies from re-training the models and from using the embedding estimator. In future research we can broaden the scope of the scenario to be able to include more methods, creating the possibility to use different methods, such as the ones discussed in Section 2.

In this implementation we have not applied normalization to the numerical attributes of the entities, which means that attributes with bigger numerical ranges can disproportionately influence the calculated distances between entities. Future research should explore the effects of normalization of the numerical attributes on the estimation quality. Future research could determine where the cut-off point for specific knowledge graphs and tasks will be and at which point it will become necessary to re-train the entire pipeline. Additional future research could explore the possibility of using the embedding estimation method instead as a measure of a *noise k* for a classification task. Where the noise k is the noise between the values of a knowledge graph and the (classification) target. The higher the value of k, the more noise exists between the data and the targets.

7. Conclusion

In this paper, we present a method for estimating knowledge graph embeddings, which estimates an embedding for a new entity in a graph for which an embedding model was already created.

The experiments show that the estimation method can result in useful estimated embeddings, which, although they have a lower accuracy, performs between 1,004-4,635 times faster. From the synthetic knowledge graph results, we conclude that the accuracy is a good indicator of how well embedding estimation method performs the estimation, since the increase of noise requires an increase of k to achieve the "best possible" accuracies. Using the Manhattan distance measure consistently resulted in the highest accuracies for the synthetic knowledge graphs, but not for the real world knowledge graphs. The creation process of the synthetic knowledge graphs and the implementation of the noise values may have contributed to this. In future research other synthetic knowledge graph benchmarks, such as GraphWorld [28], can provide more insight into the effect of distance measure selection.

Acknowledgments

This work is part of the InterConnect project (interconnectproject.eu/) which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 857237. Additionally, this work is also part of HEDGE-IoT (https://hedgeiot.eu) which has received funding from the European Union's Horizon Europe research and innovation program under grant agreement No 101136216.

References

[1] P. Ristoski, J. Rosati, T. Di Noia, R. De Leone, H. Paulheim, Rdf2vec: Rdf graph embeddings and their applications, Semantic Web 10 (2019) 721–752. doi:10.3233/SW-180317.

[2] W. L. Hamilton, R. Ying, J. Leskovec, Representation learning on graphs: Methods and applications, arXiv preprint arXiv:1709.05584 (2017).

[3] B. Steenwinckel, G. Vandewiele, T. Agozzino, F. Ongenae, pyrdf2vec: A python implementation and extension of rdf2vec, in: The Semantic Web: ESWC 2023, Lecture Notes in Computer Science, volume 13870, Springer, 2023, pp. 471–483. doi:10.1007/978-3-031-33455-9_28.

[4] V. de Boer, J. Wielemaker, J. van Gent, M. Oosterbroek, M. Hildebrand, A. Isaac, J. van Ossenbruggen, G. Schreiber, Amsterdam museum linked open data, Semantic Web 4 (2013) 237–243. doi:10.3233/SW-2012-0074.

[5] R. van der Weerdt, V. de Boer, L. Daniele, R. Siebes, F. van Harmelen, Evaluating the effect of semantic enrichment on entity embeddings of iot knowledge graphs, Proceedings of the 1st International Workshop on Semantic Web on Constrained Things at ESWC 2023. CEUR Workshop Proceedings 3412 (2023). URL: https://ceur-ws.org/Vol-3412/paper5.pdf.

[6] R. Biswas, L.-A. Kaffee, M. Cochez, S. Dumbrava, T. E. Jendal, M. Lissandrini, V. Lopez, E. L. Mencía, H. Paulheim, H. Sack, E. K. Vakaj, G. de Melo, Knowledge Graph Embeddings: Open Challenges and Opportunities, Transactions on Graph Data and Knowledge 1 (2023) 4:1–4:32. doi:10.4230/TGDK.1.1.4.

[7] R. Verdecchia, J. Sallou, L. Cruz, A systematic review of green ai, WIREs Data Mining and Knowledge Discovery 13 (2023) e1507. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1507. doi:https://doi.org/10.1002/widm.1507. arXiv:https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1507.

[8] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al., Knowledge graphs, ACM Computing Surveys 54 (2021) 1–37. doi:10.1145/3447772.

[9] F. Manessi, A. Rozza, M. Manzo, Dynamic graph convolutional networks, Pattern Recognition 97 (2020). doi:10.1016/j.patcog.2019.107000.

[10] J. Jung, J. Jung, U. Kang, Learning to walk across time for interpretable temporal knowledge graph completion, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 786–795. doi:10.1145/3447548.3467292.

[11] C. Martín, P. Langendoerfer, P. S. Zarrin, M. Díaz, B. Rubio, Kafka-ml: Connecting the data stream with ml/ai frameworks, Future Generation Computer Systems 126 (2022) 15–33. doi:10.1016/j.future.2021.07.037.

[12] N. Hubert, P. Monnin, H. Paulheim, Beyond transduction: A survey on inductive, few shot, and zero shot link prediction in knowledge graphs, arXiv preprint (2023). doi:arXiv:2312.04997.

[13] B. Shi, T. Weninger, Open-world knowledge graph completion, Proceedings of the AAAI conference on artificial intelligence 32 (2018). doi:10.1609/aaai.v32i1.11535.

[14] G. A. Gesese, H. Sack, M. Alam, Raild: Towards leveraging relation features for inductive link prediction in knowledge graphs, in: Proceedings of the 11th International Joint Conference on Knowledge Graphs, 2022, pp. 82–90. doi:10.1145/3579051.3579066.

[15] M. Galkin, X. Yuan, H. Mostafa, J. Tang, Z. Zhu, Towards foundation models for knowledge graph reasoning, The Twelfth International Conference on Learning Representations (2023). doi:10.48550/arXiv.2310.04562.

[16] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, Advances in neural information processing systems 30 (2017). doi:10.48550/arXiv.1706.02216.

[17] T. Wu, A. Khan, M. Yong, G. Qi, M. Wang, Efficiently embedding dynamic knowledge graphs, Knowledge-Based Systems 250 (2022) 109124. doi:10.1016/j.knosys.2022.109124.

[18] Y. Cui, Y. Wang, Z. Sun, W. Liu, Y. Jiang, K. Han, W. Hu, Lifelong embedding learning and transfer for growing knowledge graphs, Proceedings of the AAAI Conference on Artificial Intelligence 37 (2023) 4217–4224. doi:10.1609/aaai.v37i4.25539.

[19] X. Wilcke, P. Bloem, V. de Boer, R. van 't Veer, F. van Harmelen, End-to-end entity classification on multimodal knowledge graphs, arXiv preprint arXiv:2003.12383 (2020). doi:10.48550/arXiv.2003.12383.

[20] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: 1st International Conference on Learning Representations, ICLR 2013, Workshop Track Proceedings, 2013. doi:10.48550/arXiv.1301.3781.

[21] S. H. Hahn, H. Paulheim, Rdf2vec embeddings for updateable knowledge graphs–reuse, don't retrain!, Extended Semantic Web Conference, Poster Track (2024). URL: https://2024.eswc-conferences.org/wp-content/uploads/2024/05/77770211.pdf.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, et al., Pytorch: An imperative style, high-performance deep learning library, Proceedings of NeurIPS. Advances in Neural Information Processing Systems 32 (2019) 8024–8035. doi:10.48550/arXiv.1912.01703.

[23] P. Bloem, X. Wilcke, L. van Berkel, V. de Boer, kgbench: A collection of knowledge graph datasets for evaluating relational and multimodal machine learning, The Semantic Web: 18th International Conference, ESWC 2021. Lecture Notes in Computer Science 12731 (2021) 614–630. doi:10.1007/978-3-030-77385-4_37.

[24] L. Daniele, F. den Hartog, J. Roes, Created in Close Interaction with the Industry: the Smart Appliances REFerence (SAREF) Ontology, in: International Workshop Formal Ontologies Meet Industries, Springer, 2015, pp. 100–112. doi:10.1007/978-3-319-21545-7_9.

[25] Open Power System Data, Data Package Household Data, Version 2020-04-15 https://data.open-power-system-data.org/household_data/2020-04-15/. (Primary data from various sources, for a complete list see URL)., 2020.

[26] R. van der Weerdt, V. de Boer, L. Daniele, B. Nouwt, R. Siebes, Making heterogeneous smart home data interoperable with the SAREF ontology, International Journal of Metadata, Semantics and Ontologies 15 (2021) 280–293. doi:10.1504/IJMSO.2021.125893.

[27] R. van der Weerdt, V. de Boer, R. Siebes, R. Groenewold, F. van Harmelen, OfficeGraph: A Knowledge Graph of Office Building IoT Measurements, in: The Semantic Web: ESWC 2024. Lecture Notes in Computer Science, volume 14665, 2024, pp. 94–109. doi:10.1007/978-3-031-60635-9_6.

[28] J. Palowitch, A. Tsitsulin, B. Mayer, B. Perozzi, Graphworld: Fake graphs bring real insights for gnns, in: Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining, 2022, pp. 3691–3701. doi:10.1145/3534678.3539203.