Large-scale knowledge graph representation learning

Marwa Badrouni, Chaker Katar, Wissem Inoubli

Large Scale Knowledge Graph Representation Learning

Marwa Badrouni[1], Chaker Katar[2], and Wissem Inoubli[3]

[1]University of Jendouba, Faculty of law, Economics and Management Sciences of Jendouba, Jendouba 8189, Tunisia.
[2]University of Jendouba, Faculty of law, Economics and Management Sciences of Jendouba, Jendouba 8189, Tunisia.
[3]Univ. Artois, CNRS, UMR 8188, Computer Science Research Institute of Lens (CRIL), F-62300 Lens, France.
E-mails: [1] marouabadrouni@gmail.com.
[2] chaker.katar@gmail.com.
[3] wissem.inoubli@univ-artois.fr.

Abstract

The knowledge graph emerges as powerful data structures that provide a deep representation and understanding of the knowledge presented in networks. In the pursuit of representation learning of the knowledge graph, entities and relationships undergo an embedding process, where they are mapped onto a vector space with reduced dimensions. These embeddings are progressively used to extract their information for a multitude of tasks in machine learning. Nevertheless, the increase data in knowledge graph has introduced a challenge, especially as knowledge graph embedding now encompass millions of nodes and billions of edges, surpassing the capacities of existing knowledge representation learning systems. In response to these challenge, this paper presents DistKGE, a distributed learning approach of knowledge graph embedding based on a new partitioning technique. In our experimental evaluation, we illustrate that the proposed approach improves the scalability of distributed knowledge graph learning with respect to graph size compared to existing methods in terms of runtime performances in the link prediction task aimed at identifying new links between entities within the knowledge graph.

Keywords: knowledge graph, knowledge graph embedding, distributed learning, translating embeddings (TransE).

1 Introduction

The Knowledge Graph (KG) serves as a powerful data structure that captures relationships among diverse entities, playing a fundamental role in modeling and understanding complex knowledge. There has been a notable resurgence of interest, as evidenced by a multitude of research articles, solutions, analyst reports, communities, and conferences dedicated to the subject. This popularity can be attributed in part to the rapid advancements in graph technology in recent years, as well as the pressing need to extract meaningful insights from data. Knowledge Graph that focus on relationships between concepts and entities are often associated with related open data projects [1]. They are also employed by social media platforms like Facebook contains over 100 billion connections among entities [2] as well as search engines like Google encompasses 18 billion statements related to 570 million entities, featuring a schema that includes 1,500 entity types and 35,000 relation types [2], and Yahoo encompasses approximately 3.5 million entities and 1.4 billion relations [2]. They also include knowledge-engines and question-answering services like WolframAlpha, Apple's Siri and Amazon Alexa.

However, to fully leverage these KG in machine learning models and data analysis tasks, it is often necessary to represent entities and relations as numerical vectors, a process known as knowledge graph embedding (KGE or known also as Low-dimensional representations).

Knowledge graph embedding is a technique that transforms elements from a knowledge graph into vector representations. Learning these embeddings is intended to make it easier to manipulate graph elements (entities and relations), which are used in tasks like entity classification, link prediction, and recommender systems [3]. Although there are a variety of models such as TransE [4], TransH [5], and ComplEx [6] available for generating embeddings and their training is time consuming or infeasible for large graphs for many reasons since there are millions of nodes, billions of edges and tens of thousands of relationships in the KGs. Training embeddings of this scale requires compute resources that far exceed the capabilities of any machine. Therfore, distributed learning of knowledge graph embedding has attracted significant interest within the research community in recent years where the challenges related to the partitioning of knowledge graphs is an important step.

Moreover, there is a growing need for techniques specifically tailored to partitioning knowledge graphs, which are complex data structures containing interconnected entities, relationships and knowledge. In this research, a gap exists in this regard with relatively few methods dedicated to knowledge graph partitioning, such as AWAPart [7] and MCS [8].

In this paper we focus on distributed learning of knowledge graph embedding. As such, we propose a new partitioning method to partition the KG into a set of sub-KGE. Subsequently, these partitions are distributed among the compute nodes for training. Finally, we apply the proposed architecture to the link prediction task.

The main contributions of this paper are summarized as follows:

- We propose a distributed learning approach for knowledge graph embedding (DistKGE). This method enhances the efficiency of learning embeddings for large knowledge graphs.
- We propose a new partitioning method (PartKG) that divides the knowledge graph into sets of sub-knowledge graphs.
- We propose distributed training to efficiently process the partitioned knowledge graph, and we experimentally evaluate its effectiveness in a link prediction task.
- We conducted experimental evaluations to assess the performance of our method on different publicly available datasets.

The remainder of this paper is structured as follows. In Section 2, we present a state of the art of knowledge graph embedding methods. Section 3 introduces the proposed approach, followed by the presentation of experiments in Section 4. Section 5 covers the experimental results. Finally, we conclude the paper in Section 6.

2 State of the art

This section explores the taxonomy of knowledge graph embedding [6]. The literature presents various approaches, broadly categorized into three groups: (i) translation-based models, (ii) Semantic matching-based models and (iii) neural network-based models.

Previous research on knowledge graph embedding has showcased various methods. Let's start by examining the translation-based models. These models employ scoring methods based on distance, approaching the task of identifying valid triples as the translation of entities via relations. It presuppose that a translation vector may capture the semantic meaning of a relationship between two entities and can be used to represent that relationship. The essential concept is to learn these vectors in such a way that the relationships between entities are preserved while representing each entity and relationship as a vector in a continuous vector space. Among well-known models for KGE are:
TransE [4] remains the most classic and the first translational model. It depicts relations and entities as vectors of the same euclidean space R^d. A relation r is understood to be a translation vector that, with minimal error, joins the head entity (h) and the tail entity (t), the scoring function as illustrated by formula (1):

$$f_r(h,t) = -\|h + r - t\|_{\ell^1/\ell^2} \tag{1}$$

Where, ℓ^1/ℓ^2 represents the norm constraints. TransH [5] (with relation-specific hyperplanes), it maps entities onto a hyperplane to tackle the challenge of embedding complex relations, and TransR [9] (with relation-specific vector spaces), it addresses the complexity of relation embedding by transforming the entities using the same matrix. In our work, we use TransE as the most representative model. TransE stands out as a widely used algorithm within the realm of knowledge graph embedding, designed to acquire low-dimensional vectors for entities and relationships.

Having observed that translation-based models employ distance-based scoring functions for assessing similarity between distinct entities and relations, semantic matching models, in contrast, utilize similarity-based scoring functions. Numerous knowledge graph embedding algorithms fall under this model, and a few of them are elucidated below.

DistMult [10] represents each entity as a vector embedding and the relationship between the head and tail entities using a diagonal matrix ($diag(.)$). It uses matrix multiplication to characterize the composition of relations. The scoring function is [11]:

$$f_r(h,t) = h^T diag(r) t = \sum_{i=0}^{d-1} [r]_i \cdot [h]_i \cdot [t]_i \tag{2}$$

ComplEx [6] [12] is an extension of the DistMult model. It presents embeddings with complex values for entities and relations, enhancing the ability to deduce antisymmetric relations and uncover potential semantic associations. In ComplEx model h, r and t are located in the complex space C^d rather than the real space. The score function is calculated as illustrated by formula (3):

$$fr(h,t) = Real(h^T diag(r) \bar{t}), \tag{3}$$

Where $\bar{t}$ represents the complex conjugate of the tail entity and $Real(.)$ denotes the real part of a complex relation.

Neural network-based models have shown predictive capabilities in diverse domains like knowledge graph-based learning and natural language processing (NLP). These models use deep neural networks to capture non-linear dependencies and interactions among entities and relationships. The ConvE and ConvKB are examples for this methods.

ConvE [13] was the first model to integrate cnn arichatecture into the knowledge base completion problem and is undeniably recognized as a significant milestone in every literature survey. CNNs consists of layers of neurons that perform convolution operations to extract important features from the input data. In contrast to fully connected neural networks, CNNs have the capability to acquire nonlinear features capable of capturing intricate relationships while utilizing a significantly reduced number of parameters [6]. The key concept of ConvE lies in the use of 2D convolutional layers to process entities and relations as matrices. The scoring function is defined as illustrated by equation (4):

$$fr(h,t) = g\left(\text{vec}\left(g\left(concat(\overline{h},\overline{r}) * w\right)\right) W\right) t \tag{4}$$

Where $concat(.,.)$ is the concatenation operator, * represents convolution and g denotes a nonlinear function.

ConvKB [14] uses 1D convolutional operations to capture global relationships and temporal attributes among entities. It is a representation of the k-dimensional embedding of every triple (h,r,t) into a matrix of three rows. The score function given by eqation (5) [6]:

$$fr(h,t) = concat(\sigma([\mathbf{h},\mathbf{r},\mathbf{t}] * \Omega))\mathbf{w} \tag{5}$$

Where Ω represents a set of filters, σ is an activation function and $concat(.,.)$ is the concatenation operator [6].

3 Proposed approach

The substantial size of the data in KGs has also become increasingly massive. For example the Facebook knowledge graph contains billions of entities and trillions of edges. Similarly, the Freebase comprises millions of entities and billions of edges. With the increased amount of data, these KGs become very large which lead not only to data storage problem but also to limited processing capacity. Although the expanding size and computational complexity associated with knowledge graph, the research community has oriented its efforts to distributed learning of large graphs (the distribution of large graph learning), where one of the technical challenges is to effectively partition this graph into enough partitions ensuring that each compute node deals with a relatively small amount of data. The goal is to balance the workload across all compute nodes. In addition, the partitioning strategy consists of partitioning the knowledge graph using the Louvain algorithm [15]. This choice is attributed to its speed and capability to generate high-quality communities [16].

The architecture that we propose, illustrated by figure 1, has been developed on a distributed cluster of nodes. Below, we discuss every component of the proposed architecture.

3.1 The proposed architecture: A distributed learning of knowledge graph embedding: DistKGE

In the proposed architecture (Global architecture figure 1) the KG is first partitioned and then partitions are distributed across the compute nodes for training. Therefore, the distributed learning of knowledge graph embedding process encompasses the following steps:

- Partition the knowledge graph into p-partitions.
  This step partitions the initial knowledge graph into a set of sub-KGE using PartKG, our new partitionng method (it is discussed in detail in 3.2).
- Distributed training.
  Once the knowledge graph is partitioned into a set of sub-KGE, these partitions are eventually distributed across a set of nodes and local learning takes place to perform a link prediction task. We describe this process over a cluster of compute nodes where each node is responsible of loading data using (SGD) and running a copy of the model (Embedding Model()). This step involves launching a worker on each node and every iteration of the training process samples a negative triplets (which is usually grounded in the assumption that the model, in its operation, does not solely generate positive triples) in its partition.
  Then when we refer to the optimization step, it typically involves simultaneously optimizing both types of triplets (negative and positive). By doing so, the embedding model improves its ability to represent relationships between entities in the

knowledge graph. This enhancement contributes to the overall quality of embeddings and the model effectiveness in accurately capturing the graph structure. Ultimately, this technique strengthens the model capacity to predict new relationships and to acquire embeddings (vector representations) for entities and relations in a manner that minimizes the distance between the embeddings of positive triplets, while simultaneously maximizing the distance between the embeddings of negative triplets (explained in detail in 3.3).

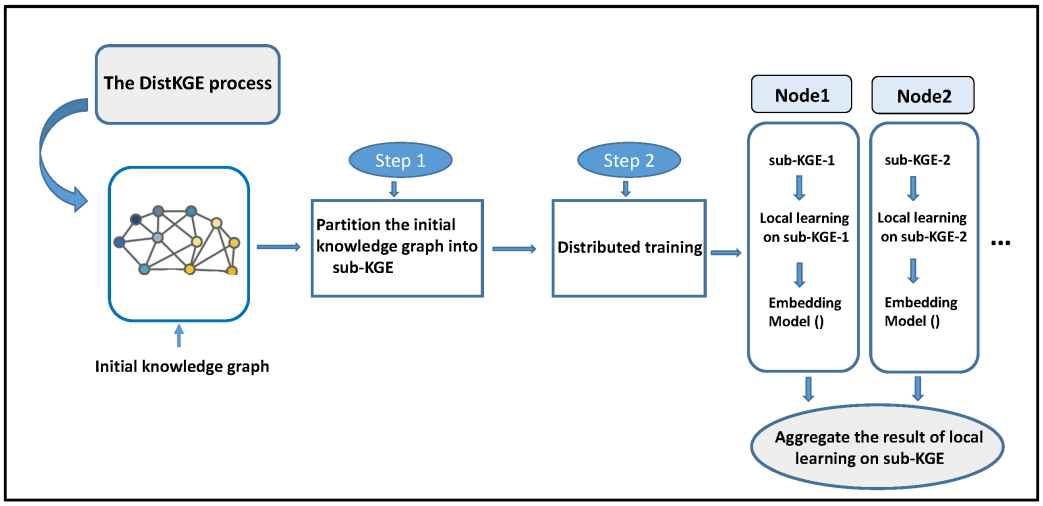

Fig. 1 Global architecture of the proposed approach: DistKGE

The following subsections provide a detailed description of each of the steps listed in the proposed architecture.

3.2 Knowledge graph partitioning

In this article, we introduce a PartKG method, which is inspired by the Louvain algorithm to partition the KG into sub-KGE. It is based on the following element such as the modularity metric [17].
Modularity[1] is defined as:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{6}$$

Where:

- Q represents modularity, which is the value we aim to maximize.

[1] Q

- A_{ij} is a binary indicator equal to 1 if a link exists between nodes i and j and 0 otherwise.
- k_i and k_j are the degrees of nodes i and j, meaning the number of links connected to these nodes.
- $2m$ is the total number of links (edges) in the graph.
- c_i and c_j represent the communities to which nodes i and j belong, respectively.
- δ is a delta function that equals 1 if nodes i and j are in the same community ($c_i = c_j$) and zero otherwise.

The main steps of PartKG algorithm

Algorithm 1 Partition KG (PartKG) summarizes the entire process. Initially, a modification is conducted to the structure of the knowledge graph to generate a suitable representation required for the execution of the Louvain algorithm [18]. Such modification results in the transformation of nodes (entities) into triplets;
($N \Leftarrow \{t_n\}$; $\forall t_n \in T$ (represents a set of triplets)) and the relations are the result of a similarity between the relations of each pair of triples;
$R \Leftarrow \{\text{Result of Sim}(r(t_i), r(t_{i+1}));\ \forall(t_i) \text{ and } (t_{i+1}) \in T\}$;
Where r represents the relation, t_i and t_{i+1} represent the number of triplets.
For this evaluation, we employ $\text{Sim} \Leftarrow \cos(T1, T2) = \left(\frac{\mathbf{T1} \cdot \mathbf{T2}}{\|\mathbf{T1}\| \|\mathbf{T2}\|}\right)$;
In the vector space model, 'Cosine similarity' is widely employed to measure the similarity between two vectors. This is due to its effectiveness in assessing the semantic proximity of relations among triplets [19]. If it achieves a high score it indicates that the pair of triplets is similar (sharing the same relation), otherwise, they are not.
If the triplets share the same relation, it indicates that this relation serves as a link between the entities in the updated knowledge graph representation. On the other hand, if the triplets do not share the same relation, a random selection is made from one of the relations within these triplets to establish a connection between the two entities.

Following this new representation of the knowledge graph, we apply the Louvain algorithm to conduct the partitioning. The latter is essentially based on the following steps:

- The first step consists of a partitioning of the graph into smaller communities.
- The second step consists of the aggregation of the communities obtained during the first step to form larger and more coherent communities.

In another perspective, it treats each triplet (entities) as an independent community;

$$C \Leftarrow ((t_n) \in T) \tag{7}$$

Where, C represents communities in the louvain algorithm, and $(t_n) \in T$ represents the triplets.
It randomly chooses a starting node and calculates the modularity Q (equation 6) of their neighbors, and this iterative procedure continues until the value of Q reaches

its maximum, this indicates that it remains constant, or until i reaches the specified value I (the algorithm stops when the number of iterations is predetermined from the beginning and is finished). Finally, an update is performed to obtain a set of sub-knowledge graphs.

Algorithm 1 PartKG algorithm

Input: Set of triplets $(t_1, \ldots, t_n)$, where $(t_1, \ldots, t_n) \in T$;
Output: KGE partition on sub-KGE;
$N \Leftarrow \{t_n \mid \forall t_n \in T\}$;
$R \Leftarrow \{\text{Result of Sim}(r(t_i), r(t_{i+1}));\ \forall(t_i) \text{ and } (t_{i+1}) \in T\}$;
$\text{Sim} \Leftarrow \cos(T1, T2) = \left(\frac{\mathbf{T1} \cdot \mathbf{T2}}{\|\mathbf{T1}\| \|\mathbf{T2}\|}\right)$
Start Louvain algorithm
$C \Leftarrow \{(t_n) \in T\}$
For $i \in (1..I)$ **Do**
  Do
    $Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m}\right] \delta(c_i, c_j)$
  While (Q reaches its maximum)
End For
Return (KGE partition on sub-KGE)

Upon completion of the partitioning process, the graph partitions identified by the PartKG algorithm are then forwarded to the distributed training phase, as detailed in the subsequent section 3.3.

3.3 Distributed training of knowledge graph embedding

The distributed processing is carried out using Ray framework which is a unified way to scale Python and AI applications from a laptop to a cluster.
With Ray, you can seamlessly scale the same code from a laptop to a cluster. Ray is designed to be general-purpose, meaning that it can performantly run any kind of workload.

Step1: Local learning on each sub-KGE. As shown in figure 1, the initial knowledge graph is fragmented into several partitions, each of them is assigned to a machine (node). Each compute node is specifically assigned to one or more sub-KGE. This assignment remains constant throughout the training phase, ensuring a continuous focus on a specific subset of data. It facilitates the development of in-depth expertise on the relationships and entities within the assigned partition for each node.

Then, a set of negative triplets is generated, composed of combinations of artificially created relations to enrich the training dataset. This process is crucial for enhancing the model ability to discern relevant relationships within the assigned partition by introducing examples of relations that may not be present in the initial knowledge graph. The next task in the distributed training process involves training the embedding model (transE) using the complete triplet set. It is crucial for enriching the training dataset by introducing more complex and varied scenarios. To achieve

this, we combine the newly generated negative with the existing positive triplets. This combination creates a diverse set that exposes the model to a variety of potential relations. Each node, based on its assigned data partition, uses this set to train the graph embedding model.

During the training process, the model adjusts its parameters using the complete triplets set. This enables it to learn from existing positive relations and comprehend potential relations between entities. Throughout each iteration, the model parameters are fine-tuned to minimize the predicted loss. This loss is assessed by comparing the model predictions with the actual relations observed in the complete training set, which includes both existing positive relations and potential relations generated during the training process. The optimization objective is to align it more closely with the real and potential relations in the dataset. This alignment enhances the model ability to generalize and make more accurate predictions.

Step2: Aggregate the result of local learning on sub-KGE. The aggregation involves combining these local learning results from all computing nodes to create a unified and comprehensive model update. It ensures that the knowledge acquired by each node during its local learning contributes to the collective understanding of the entire knowledge graph. It contributes to building a globally informed model that reflects the collective knowledge acquired from different data.

This step is designed to construct a model that efficiently captures the richness and diversity of data associated with this type of graph. Consequently, it enhances the model ability to comprehend more intricate relationships within the large knowledge graph.

To validate our contribution we carry out the different experiments in the following section.

4 Experiments

In our experiments, we tested our proposed approach using MalKG a malware dataset of cybersecurity domain. The main objective behind these was to predict rates of new relationships among these malware instances using knowledge graph embedding (KGE). This approach assists cybersecurity managers in discovering and predicting future malware. Aditionnaly, we have experimented our distributed learning approach using datasets of other domains including FB15K, WN18, and FB15K-237 as illustrated in table 1.

Table 1 Dataset statistics

Dataset	Entities	Relations	Triples
FB15K	14.951	1345	592.213
MalKG	27.354	34	40.000
WN18	40.943	18	151.442
FB15K-237	14.541	237	310.116

- FB15K dataset [20]: Freebase is a public dataset knowledge graph, it contains a very large set of data in different domains like film, music, medicine, business etc. There are presently more than 80 million entities and 1.2 billion triplets. This version includes 592.213 triplets with 14,951 entities and 1,345 relationships.
- MalKG dataset [21]: The malware knowledge graph or MalKG represents heterogeneous multi-modal data on malware in the cybersecurity domain. This dataset consists of almost 40.000 triples, produced by 27.354 entities and 34 relations.
- WN18 dataset [22]: The WordNet knowledge graph consists of a subset of relations from the WordNet lexical database. It contains 151.442 triples, 18 lexical and semantic relations and 40.943 entities.
- FB15K-237 dataset [23]: Is a subset of FB15K which contain 310.116 triplets with 14.541 entities and 237 relationships.

Link Prediction

The link prediction (LP) task refers to the process of utilizing the existing triplets within a knowledge graph to deduce the new ones. This essentially involves predicting the correct entity that completes a valid triple with ⟨h, r, ?⟩ or ⟨?, r, t⟩ [24] in order to forecast the new relationship. We divide the set of triplets T into three subsets: train, valid and test as shown in table 2.

Table 2 Number of training, validation and testing triples on each dataset. Each split includes positive and negative triples

Dataset	Training triples	Validation Triples	Testing Triples
FB15K	483.142	50000	59071
MalKG	28.000	6000	6000
WN18	141.442	5000	5000
FB15K-237	272.115	17535	20466

Metrics

To assess the performance of our model against baseline methods. In link prediction, we use Mean Rank, Hits@10 (the proportion of correct entities ranked in the top 10) and Hits@5 as a metrics.

Mean Rank

Corresponds to the arithmetic mean over rank Scores of the triples:

$$Mean\ Rank = \frac{1}{|P|} \sum_{r \in P} r \tag{8}$$

Hits@10

Is a metric that accounts for the proportion appearing in the first 10 top-scored triples.

This metric is bounded in the [0, 1] range and its values increases with $[0, 1]$:

$$Hits@10 = \frac{(|r \in P|r \leq 10)}{|P|} \tag{9}$$

where $|P|$ is the number of rank scores and r is rank.

Hits@5

Is a metric that accounts for the proportion appearing in the first 5 top-scored triples. This metric is bounded in the [0, 1] range and its values increases with $[0, 1]$:

$$Hits@5 = \frac{(|r \in P|r \leq 5)}{|P|} \tag{10}$$

where $|P|$ is the number of rank scores and r is rank.

System Setup

Our proposed architecture was developped on Ray 2.3.1 [25] as the distributed training framework, PyTorch 1.9.0 [26] as the backend deep learning framework and PyTorch Geometric 1.7.2 [27] for graph embedding. Our experiments were conducted on a cluster consisting of three nodes. We used transE model to learn the embeddings of nodes and relationships. For this model, we set the hyperparameters as follows: learning rate 0.005, embedding dimension K = 5. For all experiments, we train all models for a maximum of 1000 epochs using Adam [28] optimizer.

5 Experimental results

In this section, the experiments are conducted to better comprehend the impact of centralized processing on the quality of knowledge graph embedding. Additionally, we aim to assess performance in terms of processing time using various metrics, including MR, Hits@10, and Hits@5, for the task of link prediction.

Table 3 Result of centralized processing with DGL-KE

	Centralized processing with DGL-KE			
Dataset	MR	Hits@10	Hits@5	Time processing (ms)
FB15K	243.26	0.354	0.265	72
MalKG	1167.76	0.130	0.094	217
WN18	6428	0.338	0.320	103
FB15K-237	477.02	0.240	0.171	59

The tables (3 and 4) shown the results of various metrics on the FB15K, MalKG, WN18 and FB15K-237 datasets using two popular library Deep Graph Learning(DGL-KE) [29] and TorchKGE [30]. We measured the processing time for the TransE model shows that really there is not a big difference between the two methods used. The results show that with the use of DGL-KE the values of the metrics are higher than TorchKG. Therefore, we observe that the TransE model gives more efficient results

Table 4 Result of centralized processing with TorchKGE

	Centralized processing with TorchKGE			
Dataset	MR	Hits@10	Hits@5	Time processing (ms)
FB15K	221	0.283	0.254	75
MalKG	1092.46	0.116	0.068	223
WN18	4207.23	0.332	0.304	107
FB15K-237	429	0.262	0.157	64

with DGL-KE in the new link prediction task. Using DGL-KE we get the best training time compared to TorchKG. On the FB15K dataset we achieve a time of 72 ms compared to 75 ms with torchKG. MalKG, WN18 and FB15K-237 are attained a time of 217 ms, 103 ms and 59 ms against 223 ms, 107 ms and 64 ms with TorchKG.

These experiments reveal the results regarding the model performance in link prediction across diverse datasets in a centralized processing setting. The metrics MR, Hits@10, and Hits@5 provide insights into the quality of the embedding, while the processing time conveys information about its efficiency and speed in accomplishing the desired task.

Results of the running time of distributed training

In this experiments, we present the outcomes of our experimental strategy. We conduct a comparison between our proposed distributed training method and a centralized training. First of all, we assess the efficacy of various partitioning strategies apart from DistKGE and their impact on execution times with random partitioning and Metis partitioning[31]. With random partitioning the knowledge graph is divided randomly, and in the context of Metis the process seeks to minimize the edge cut between partitions, balance the workload among them and obtain high-quality partitions.

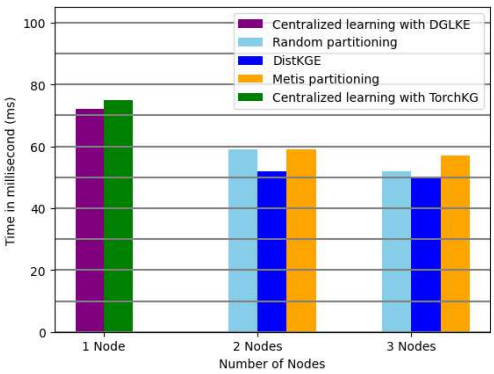

Fig. 2 Results of training time on the FB15K dataset

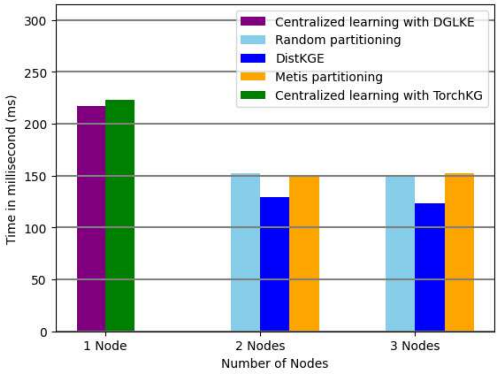

Fig. 3 Results of training time on the MalKG dataset

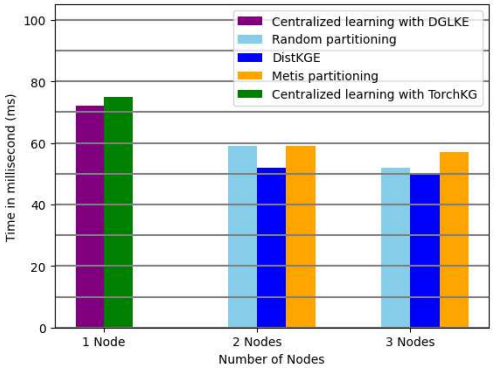

Fig. 4 Results of training time on the WN18 dataset

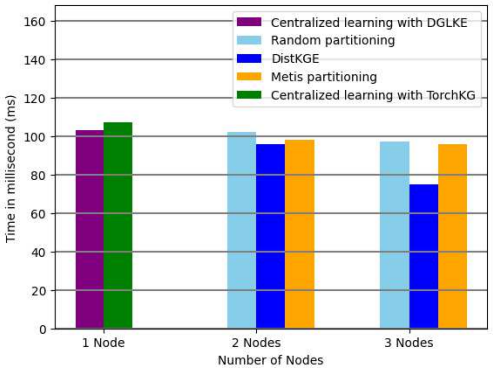

Fig. 5 Results of training time on the FB15K-237 dataset

In figures 2, 3, 4, and 5, the comparison of DistKGE performance in link prediction tasks across various datasets reveals that it outperforms other methods in terms of processing time. Specifically, on the FB15K dataset, utilizing 2 nodes and 3 nodes with DistKGE results in the best training times 52 ms and 50 ms, respectively, surpassing the random partitioning method 59 ms and 52 ms on 3 nodes, and the Metis partitioning method 59 ms and 57 ms. Moving to the MalKG dataset, DistKGE demonstrates efficient processing times of 128 ms and 123 ms on 2 and 3 nodes, respectively, compared to random partitioning 152 ms and 149 ms, and Metis partitioning 149 ms and 152 ms. The WN18 dataset exhibits a similar trend, with DistKGE achieving times of 96 ms and 75 ms on 2 and 3 nodes, while random partitioning and Metis partitioning show higher times, ranging from 97 ms to 102 ms. Finally, for the FB15K-237 dataset, DistKGE delivers impressive processing times of 56 ms and 50 ms on 2 and 3 nodes, outperforming random partitioning 58 ms and 56 ms, and Metis partitioning 58 ms on both nodes.

We can conclude that our DistKGE approach gives better results in the new link prediction task than random partitioning and metis. As well as when increasing the number of nodes the processing will be carried out in a minimum of time so it is more accelerated than working on a single node.

Evolution of the MR metric in the different approaches

To evaluate the performance of our model against random partitioning and Metis partitioning . We vary the number of training 1, 2 and 3 nodes.
Figure 6 illustrates that our proposed approach outperforms Metis, and in comparison to random partitioning, demonstrates highly efficient results across various datasets, including FB15K, MalKG, WN18, and FB15K-237. The distributed training, 2 and 3 Nodes, MR scores were more scalable compared to centralized processing (1 node). We observe that with 2 nodes we obtain the best score of 7796.3 from the WN18 dataset for the DistKGE method.

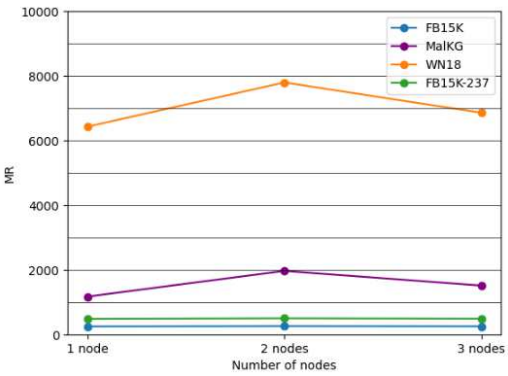

Fig. 6 Evolution of MR in DistKGE method

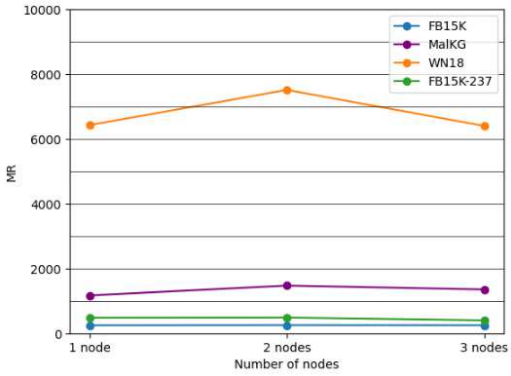

Fig. 7 Evolution of MR in random method

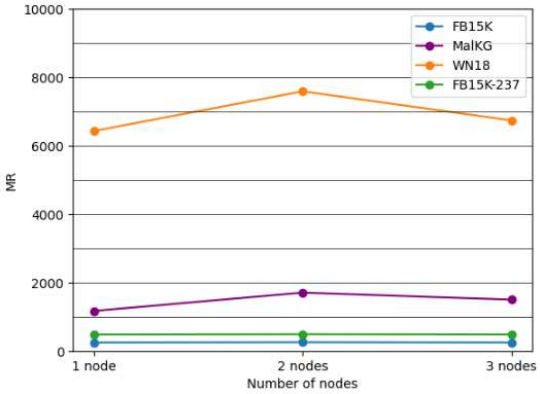

Fig. 8 Evolution of MR in metis partitioning method

Therefore working with a distributed setup offers significant advantages in terms of embedding quality efficiency and processing time (as its shown at results of the running time of distributed training).

Evolution of the Hits@10 metric in the different approaches

In these experiments, figures 9, 10, and 11, we compare the performance of the embedding quality using the Hits@10 metric in the link prediction task of all knowledge graph partitioning methods, DistKGE, random partitioning and Metis partitioning. We note the same observation that working in distributed setup gives better results than a centralized processing (1 node), for example for the FB15K dataset with the DistKGE method at a score of 41% on 2 nodes and 38% on 3 nodes while in centralized mode it reaches a score of 35%. As well as for the DistKGE approach, the scores of this metric achieve more efficient values than those of other methods.

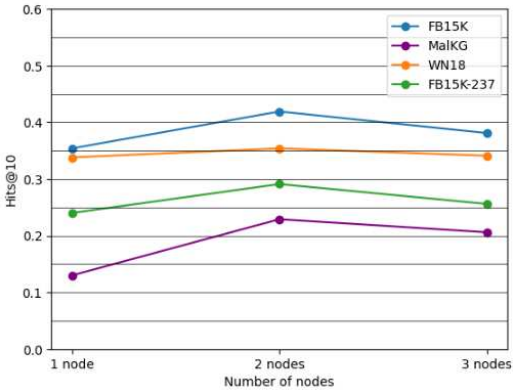

Fig. 9 Evolution of Hits@10 in DistKGE method

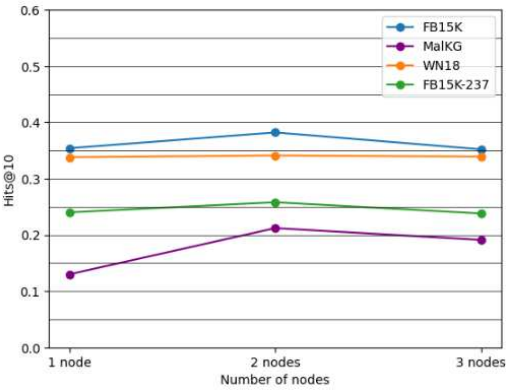

Fig. 10 Evolution of Hits@10 in random method

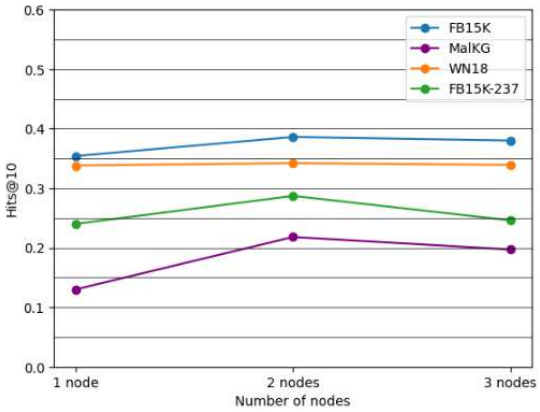

Fig. 11 Evolution of Hits@10 in metis partitioning method

Therefore, we can deduce that when we do well-guided distributed learning we can have effective results in terms of model performance than doing random learning.

Evolution of the Hits@5 metric in the different approaches

The figures 12, 13 and 14 show that our proposed approach DistKGE is scalable compared to centralized training, which takes much longer during this experience. For example the WN18 dataset has a score of 34% and 33% with DistKGE, 33% and 32% for random partitioning and 34% and 33% for Metis partitioning, while in centralized mode we have a score of 32% . And almost it is the same case for the following datasets FB15K, MalKG, and FB15K-237 the score of the Hits@5 metric realizes an evolution from 1 node to 2 and 3 nodes.

We can therefore deduce that when we do well-guided distributed learning we can have effective results in terms of model performance than doing random learning.

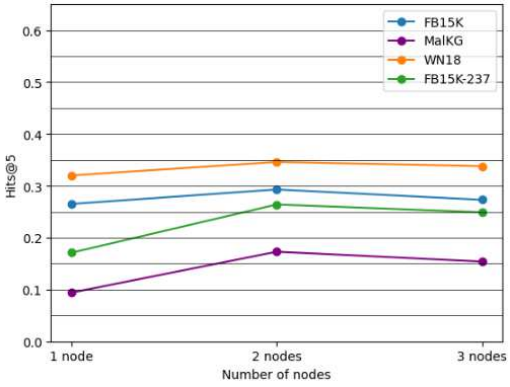

Fig. 12 Evolution of Hits@5 in DistKGE method

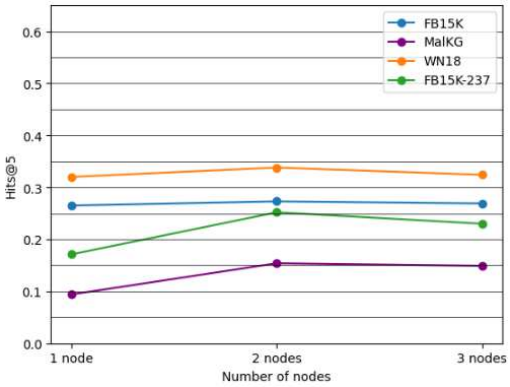

Fig. 13 Evolution of Hits@5 in random method

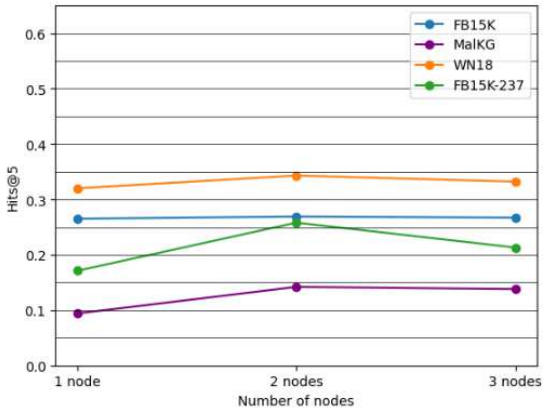

Fig. 14 Evolution of Hits@5 in metis partitioning method

6 Conclusion and future works

In this paper, we proposed DistKGE, a distributed learning approach for knowledge graph embedding. DistKGE is based on a PartKG, a novel approach for partitioning knowledge graph using the Louvain algorithm. Subsequently, we establish distributed training with Ray for the TransE knowledge embedding model. We compared our proposed method for a link prediction task. Our experimental results shown that working in centralized system does not affect the processing efficiency of a task. While using our contribution shows processing speedup on a set of nodes and our proposed algorithm is scalable than non-distributed training.

In our future work, we will attempt to alter the partitioning method and assess its impact comparing it with our own approach for example using other partitioning algorithms and also distributed processing on more nodes is a very interesting task to test it.

References

[1] Soylu, A., Corcho, O., Elvesæter, B., Badenes-Olmedo, C., Martínez, F.Y., Kovacic, M., Posinkovic, M., Makgill, I., Taggart, C., Simperl, E., *et al.*: Enhancing public procurement in the european union through constructing and exploiting an integrated knowledge graph. In: The Semantic Web–ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part II 19, pp. 430–446 (2020). Springer

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

[3] Zhang, Z., Guan, Z., Zhang, F., Zhuang, F., An, Z., Wang, F., Xu, Y.: Weighted knowledge graph embedding. In: Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 867–877 (2023)

[4] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. Advances in neural information processing systems **26** (2013)

[5] Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 28 (2014)

[6] Ferrari, I., Frisoni, G., Italiani, P., Moro, G., Sartori, C.: Comprehensive analysis of knowledge graph embedding techniques benchmarked on link prediction. Electronics **11**(23), 3866 (2022)

[7] Priyadarshi, A., Kochut, K.J.: Awapart: Adaptive workload-aware partitioning of knowledge graphs. arXiv preprint arXiv:2203.14884 (2022)

[8] Zhong, J., Wang, C., Li, Q., Li, Q.: A new graph-partitioning algorithm for large-scale knowledge graph. In: International Conference on Advanced Data Mining and Applications, pp. 434–444 (2018). Springer

[9] Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29 (2015)

[10] Li, Z., Ji, J., Fu, Z., Ge, Y., Xu, S., Chen, C., Zhang, Y.: Efficient non-sampling knowledge graph embedding. In: Proceedings of the Web Conference 2021, pp. 1727–1736 (2021)

[11] Choudhary, S., Luthra, T., Mittal, A., Singh, R.: A survey of knowledge graph embedding and their applications. arXiv preprint arXiv:2107.07842 (2021)

[12] Lv, X., Hou, L., Li, J., Liu, Z.: Differentiating concepts and instances for knowledge graph embedding. arXiv preprint arXiv:1811.04588 (2018)

[13] Dettmers, T., Minervini, P., Stenetorp, P., Riedel, S.: Convolutional 2d knowledge graph embeddings. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)

[14] Nguyen, D.Q., Nguyen, T.D., Nguyen, D.Q., Phung, D.: A novel embedding model for knowledge base completion based on convolutional neural network. arXiv preprint arXiv:1712.02121 (2017)

[15] Zhang, J., Fei, J., Song, X., Feng, J.: An improved louvain algorithm for community detection. Mathematical Problems in Engineering **2021**, 1–14 (2021)

[16] Ghosh, S., Halappanavar, M., Tumeo, A., Kalyanaraman, A., Lu, H., Chavarria-Miranda, D., Khan, A., Gebremedhin, A.: Distributed louvain algorithm for graph community detection. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 885–895 (2018). IEEE

[17] Chen, M., Kuzmin, K., Szymanski, B.K.: Community detection via maximization of modularity and its variants. IEEE Transactions on Computational Social Systems **1**(1), 46–65 (2014)

[18] Wickramasinghe, A.N., Muthukumarana, S.: Social network analysis and community detection on spread of covid-19. Model Assisted Statistics and Applications **16**(1), 37–52 (2021)

[19] Li, B., Han, L.: Distance weighted cosine similarity measure for text classification. In: Intelligent Data Engineering and Automated Learning–IDEAL 2013: 14th International Conference, IDEAL 2013, Hefei, China, October 20-23, 2013. Proceedings 14, pp. 611–618 (2013). Springer

[20] Bast, H., Bäurle, F., Buchhold, B., Haußmann, E.: Easy access to the freebase dataset. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 95–98 (2014)

[21] Rastogi, N., Dutta, S., Christian, R., Zaki, M., Gittens, A., Aggarwal, C.: Information prediction using knowledge graphs for contextual malware threat intelligence. arXiv preprint arXiv:2102.05571 (2021)

[22] Kotnis, B., Nastase, V.: Analysis of the impact of negative sampling on link prediction in knowledge graphs. arXiv preprint arXiv:1708.06816 (2017)

[23] Che, F., Zhang, D., Tao, J., Niu, M., Zhao, B.: Parame: Regarding neural network parameters as relation embeddings for knowledge graph completion. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 2774–2781 (2020)

[24] Wang, M., Qiu, L., Wang, X.: A survey on knowledge graph embeddings for link prediction. Symmetry **13**(3), 485 (2021)

[25] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M.I., *et al.*: Ray: A distributed framework for emerging {AI} applications. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 561–577 (2018)

[26] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)

[27] Fey, M., Lenssen, J.E.: Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428 (2019)

[28] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[29] Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., Xiong, H., Zhang, Z., Karypis, G.: Dgl-ke: Training knowledge graph embeddings at scale. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 739–748 (2020)

[30] Boschin, A.: Torchkge: Knowledge graph embedding in python and pytorch. arXiv preprint arXiv:2009.02963 (2020)

[31] Liu, H., Zhao, M., Zhang, C., Fu, G.: Comparing topological partitioning methods for district metered areas in the water distribution network. Water **10**(4), 368 (2018)