Scaling Knowledge Graph Embedding Models for Link Prediction

Nasrullah Sheikh
IBM Research Almaden
San Jose, California, US
nasrullah.sheikh@ibm.com

Xiao Qin
IBM Research Almaden
San Jose, California, US
xiao.qin@ibm.com

Berthold Reinwald
IBM Research Almaden
San Jose, California, US
reinwald@us.ibm.com

Chuan Lei
Instacart
San Francisco, California, US
chuan.lei@instacart.com

Abstract

Developing scalable solutions for training Graph Neural Networks (GNNs) for link prediction tasks is challenging due to the inherent data dependencies which entail high computational costs and a huge memory footprint. We propose a new method for scaling training of knowledge graph embedding models for link prediction to address these challenges. Towards this end, we propose the following algorithmic strategies: self-sufficient partitions, constraint-based negative sampling, and edge mini-batch training. The experimental evaluation shows that our scaling solution for GNN-based knowledge graph embedding models achieves a 16x speed up on benchmark datasets while maintaining a comparable model performance to non-distributed methods on standard metrics.

CCS Concepts: • **Computing methodologies** → Distributed algorithms; Learning latent representations; *Knowledge representation and reasoning*.

Keywords: Knowledge Graph Embedding, Graph Neural Networks, Distributed Data Parallel Training

ACM Reference Format:
Nasrullah Sheikh, Xiao Qin, Berthold Reinwald, and Chuan Lei. 2022. Scaling Knowledge Graph Embedding Models for Link Prediction. In *2nd European Workshop on Machine Learning and Systems (EuroMLSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3517207.3526974

EuroMLSys '22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9254-9/22/04…$15.00
https://doi.org/10.1145/3517207.3526974

1 Introduction

Graphs are widely used to model and manage relational data [28]. Knowledge graphs (KG), as a prime example of graphs, model real-world objects, events, and concepts as well as various relations among them. Representation learning on large-scale knowledge graphs has been emerging as a pivotal tool to derive insights from graph structured data powering a wide range of applications such as data integration [23, 27, 30] and question answering [1, 8].

KG embedding methods [2, 3, 13, 18, 21, 26, 32] capture the attributes of entities and structures of relations in KGs, and project them into a lower dimensional vector space for use in various downstream tasks such as node classification [21] and link prediction [2, 18, 21, 22, 29]. Traditional KG embedding methods learn various patterns between the entities such as *symmetric*, *anti-symmetric* and *inverse* relations. They mainly focus on the scoring aspect of the problem, which is to predict the legitimacy between two entities and a particular relation type. Recently, message passing-based graph neural networks (GNNs) have been adopted to improve the expressive power of entity[1] embeddings [21]. GNNs capture the topological features of the entities such as shapes of the neighborhood sub-graphs which are overlooked by the traditional KG embedding methods. However, using GNNs to learning better embeddings comes at the cost of increased model complexity in terms of number of trainable parameters. For example, TransE[2] has 1.5 million parameters while RGCN[21] has 3.3 million parameters on the FB15k-237 dataset with an embedding size of 100. The increased number of trainable parameters leads to an increase in training time.

Besides model complexity, the size of modern input KGs has also grown exponentially. The Facebook Graph [4] has billions of vertices and trillions of edges, and Freebase [6] has millions of entities and billions of edges. Iterative training on these large graphs may not be feasible on single node systems due to its high computational cost and its high memory requirements.

[1]Entity and vertex terms are used interchangeably.

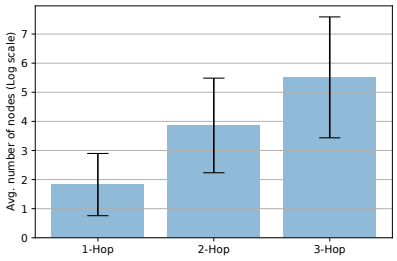

Figure 1. Average number of vertices required to compute the embedding of a vertex in ogbl-citation2 dataset.

Various distributed training frameworks [12, 34, 35] have been proposed to scale KG embedding methods. However, these frameworks only apply to the traditional models with mutually independent training triplets [2, 29]. The input data can be partitioned easily, and the models are subsequently trained in parallel. These frameworks cannot be used for training GNN-based KG embedding models [18, 21] due to the inherent dependencies in the neighborhood information (usually beyond n-hop with $n \geq 2$). Figure 1 shows that with larger and deeper neighborhoods, the average number of vertices required to compute an embedding rises significantly, which consequently leads to an increase in the number of model parameters with more computation cost and higher memory footprint. We further observe that the skewed distribution of vertex degrees in enterprise knowledge graphs leads to vertex dependencies up to tens of thousands of vertices. These dependencies make scaling GNN-based KG embedding models extremely challenging.

Several distributed GNN training frameworks have been proposed primarily for node classification [14, 33]. Graph partitioning followed by distributed training are commonly explored by these solutions. While a simple partitioning strategy is to partition the graph using either vertex-cut or edge-cut-based methods, and access required dependent vertices in other partitions remotely during training. However, the increase of the number of GNN layers, i.e., the number of hops, increases the number of messages with neighborhood information that are exchanged across partitions, which leads to a significant communication overhead. It is in contrast to distributed neural network training on non-graph structured datasets such as images, which only incurs communication overhead due to sharing of gradients. The exchange of neighborhood information is the main bottleneck in scaling GNN training. The challenge is to generate optimized graph partitions that reduce the required exchange of neighborhood information. Moreover, partitions generated from larger graphs are of considerable size and cannot fit into the smaller memory of a GPU for hardware acceleration. DistDGL [33] introduces an edge-cut-based partitioning method using METIS [10], and employs a mini-batch training approach for node classification. Edge-cut-based methods produce partitions with edge replication in multiple partitions, which may lead to skewed partition sizes. The larger partitions will be the stragglers in the training process. We observe that partitions produced by METIS followed by neighborhood expansion for link prediction are approximately 33% larger than the partitions produced by vertex-cut based methods [19], which increases the training time by approximately 21% and makes the approach sub optimal for link prediction.

In this paper, we propose a distributed training approach for GNN-based knowledge graph embedding models for link prediction. We introduce a vertex-cut method to partition the graph and then expand the partitions to include n-hop neighbors, where n is determined by the number of convolutional layers of the GNN model. The partitions produced are self-sufficient and thus do not require any exchange of neighborhood information during distributed training at the expense of data replication and redundant computation. We generate negative samples within the partitions to further reduce the communication overhead. Using a data parallel approach, we train the model in a cluster where each trainer process trains on a partition using an edge mini-batch training strategy. Our main contributions are as follows.

- To the best of our knowledge, we propose the first architecture for distributed GNN-based knowledge graph embedding model training for link prediction. We also introduce edge mini-batch training which allows us to train on large partitions.
- We employ a vertex-cut-based partitioning strategy that partitions the graph into sets of disjoint edges, which we then expand to self-contained graph partitions by replicating n-hop dependent vertices and edges required for message passing.
- We exploit the locally closed world assumption [2, 24] and employ a constraint based negative sampling strategy to sample negative samples. The negative samples are drawn from within the partition to avoid communication overhead.
- We experimentally evaluated the performance of our proposed system on two public datasets. Our approach achieves a speedup of 16x with 8 trainers without any loss on the measured metrics.

The rest of the paper is organized as follows: Section 2 describes our system architecture. In Section 3, we describe the system setup for evaluation, followed by a discussion of the results. Finally, we conclude in Section 4.

2 Distributed Knowledge Graph Training

In this section, we describe our distributed learning process of GNN-based KG embedding models on a cluster of compute nodes with multi CPUs/GPUs.

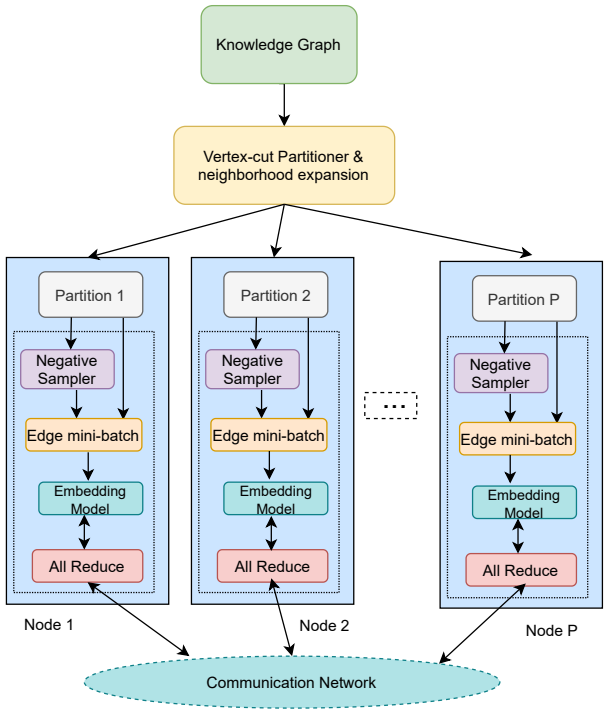

Figure 2. Architecture of our distributed GNN training approach for link prediction.

2.1 System Overview

The architecture[2] of our proposed approach is shown in Figure 2. Our proposed architecture is designed to run on a distributed CPU/GPU cluster. Each compute node (CPU/GPU) in a cluster runs a replica of the model and is responsible for training on a partition of data using synchronous gradient descent (SGD). Each training process computes the gradients of the model on an edge mini-batch, shares and averages the gradients, and updates the local model. Specifically, our distributed KG embedding learning involves the following steps:

1. Partition the graph into P disjoint subsets, and then expand each partition to include n-hops of neighbors of each vertex in the partition, where n is determined by the number of graph convolutional layers in the embedding model. The number of partitions is equal to the number of compute nodes available. We refer to compute node as a processing unit (CPU/GPU). Graph partitions along with the required features of vertices are assigned to a compute node.
2. One training process/worker is launched per compute node. During each epoch, each training process samples s negative samples for each positive sample in its partition. The number of training examples in a partition is $p \times (s+1)$, where p is the number of positive samples in a partition.
3. Each training process implements edge mini batching for training. A batch of b edges (positive and negative) in a partition is sampled. Edge mini-batch ensures that the embedding of all entities required for scoring the edges in the edge mini-batch is computed.
4. After the formation of an edge mini-batch, a computational graph is generated for message passing in the graph convolutional layers. We obtain the loss, and compute the gradients. The gradients are shared using AllReduce, and the model is optimized based on the averaged gradients. Our proposed approach can be applied to any graph embedding model which uses a message passing approach for graph convolution.

The enumerated steps in the overview are described in detail in the following subsections.

[2]For the purpose of the overview, we assume one compute node runs only one worker.

2.2 Graph Partitioning

Partitioning the input graph is an important preprocessing step in distributed training. The quality of partitions have a direct impact on the learned model quality and on scalability. We apply a two-phase approach by first partitioning the graph, and then performing neighborhood expansion to make the partitions self sufficient.

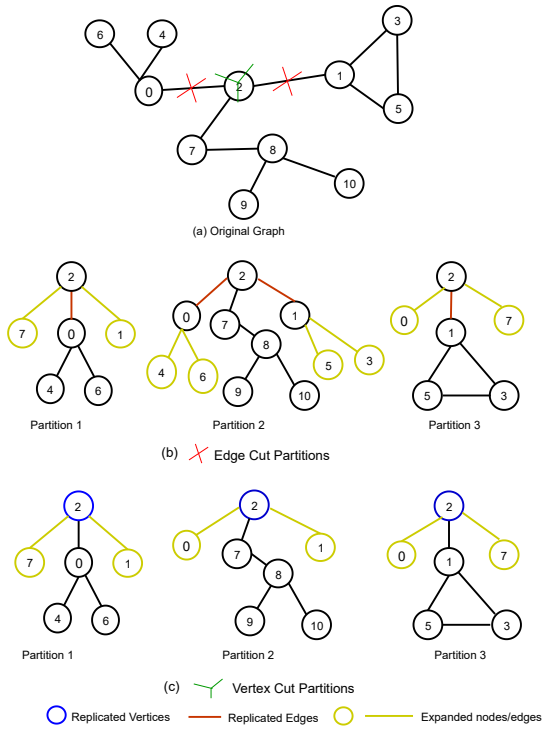

Figure 3. Graph partitioning: edge cut and vertex cut partitions along with their neighborhood expansion.

2.2.1 Partitioning. Using edge-cut partitioning methods such as Metis[10], some positive edges are replicated in multiple partitions as shown in Fig 3(b) (e.g., Edges $(0, 2)$, $(1, 2)$ are present in all partitions). Hence, the training on the replicated edges is repeated in multiple partitions which incurs additional computational cost, and may also negatively impact the learning process. Moreover, edge-cut partitioning is shown to be ineffective in balancing the workload of large real-world graphs [7, 31]. This load imbalance leads to a substantial stalling of work which increases the overall training time.

Vertex cut partitioning [7, 20, 31] divides the edges into disjoint partitions and produces balanced partitions by minimizing the vertex replication. We refer to edges in a partition as *core edges*, the vertices where the graph is partitioned as *replicated-vertices*, and other vertices as *core-vertices*. The set of *core edges* form the positive edges for training. The disjoint partitions produced by vertex-cut partitions are more suited for the problem of link-prediction because the produced partitions are balanced and neighborhood expansion does not lead to graph explosion (discussed in detail in 2.2.2).

2.2.2 Neighborhood Expansion. Link prediction requires an updated embeddings of the vertices of an edge to calculate the score that determines the validity of an edge. To compute an embedding of a vertex, a n layer GNN requires to have the features from the n-hop neighbors. Due to partitioning, some edges will have partial neighborhood information available within the partition, and the other required information could be present in different partitions. We call these edges *boundary-edges*. One possible way is to fetch this information during training. But this would incur recurring communication cost resulting in extensive training time. We propose to make the partitions independent by including the missing partial neighborhood information of *boundary-edges* in each partition. We call this process *neighborhood expansion*, and it is done after creating the partitions. Neighborhood expansion eliminates the communication cost of fetching data from other partitions, but at the expense of increasing the size of each partition. We refer to the added vertices and edges as *support-vertices* and *support-edges* respectively. Neighborhood expansion of the graph is shown in Figures 3(b) and (c). Figure 3(b) demonstrates that edge-cut partitions explode during neighborhood expansion thus limiting its usability.

2.3 Training

Each compute node will have a replica of the graph embedding model, and will work on a single graph partition. The partition assigned to a compute node remains fixed during the entire training process. We generate a set of negative edges, and the combined set of negative edges and *core edges* form the set of training edges. We employ an edge mini-batch training approach to train the KG embedding methods. The distributed training process is shown in Algorithm 1.

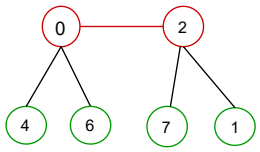

Figure 4. Edge Mini-batch

2.3.1 Negative Sampling. In KG embedding methods, negative samplers generally exploit the closed world hypothesis which considers any edge not explicitly present in the graph as a negative example. Most of the negative samples generated by this result are easy negative examples [11]. The error gradients from these samples are very small, hence do not help in learning a good model. The negative samples space $O(N^2)$ is far larger than the positive samples space, and thus it is more prone to generate easy negative samples.

We propose a constraint-based negative sampling approach. Our proposed approach considers the core edges in a partition as the positive samples, and each partition is independent of other partitions. We employ a constraint that generates the negative samples from the core vertices of the partition based on the local world hypothesis. This constraint provides two advantages, 1) the embeddings of entities in negative samples are not stale, and 2) it avoids the communication cost of querying other partitions and fetching data. This also reduces the sample space of negative samples as $N_i << N$, where N_i number of vertices in the i-th partition and N is the total number of vertices in a KG, respectively. This helps in reducing the problem of generating easy negative samples.

2.3.2 Edge Mini-Batch. GNN training on a large dataset for node classification is done by mini-batching. In mini-batching, a set of vertices is randomly selected, and a computational sub-graph is sampled for training to obtain the embeddings of the selected vertices. Using a vertex sampling strategy for link prediction is not trivial as it does not guarantee that both vertices of an edge are in the sample. For this reason, we are proposing to use *edge mini-batching* for link prediction. In *edge mini-batching*, a batch of edges is sampled, and the vertices in the batch form a vertex set. Then, a computational graph for message passing is created which captures the n-hop dependencies of the sampled edge batch. The embeddings are learned for the vertices in the vertex set. Figure 4 shows a 1-hop computational graph for an edge $(0, 2)$, and message passing is done on this graph to learn the embeddings of vertex 0 and 2.

2.3.3 Model Training. For each edge mini-batch training, we generate a computational graph from the partitioned data using the vertices in the graph. The computational graph

is used for generating embeddings for the vertices in the edge mini-batch using the graph convolutional layers of the KG embedding model. This constitutes the forward pass of the training. In the next step, a loss is calculated for the set of edges in the mini-batch which subsequently generates gradients. The gradients are shared among the training processes using the AllReduce communication primitive. Once the gradients are shared and averaged, the training processes update their local model.

Algorithm 1: Training Process on each compute node

Input: GNNmodel, optimizer, features, gPartition, epochs

```
epoch ← 1
while epoch ≤ epochs do
    negativeEdges ← negativeSampler(gPartition)
    while batch ∈ batches(negativeEdges, gPartition) do
        cgraph ← getComputeGraph(batch, gPartition)
        embedding ← GNNmodel(features, cgraph)
        loss ← loss(embedding, batch)
        backward(loss) ;      /* Gradients are computed & shared */
        optimizer.step() ;    /* The model is updated */
    end
    epoch ← epoch + 1
end
```

3 Experimental Evaluation

3.1 System Setup

We ran our experiments on a cluster of 4 machines. Each node has two Intel Xeon 6138 CPUs @ 2.00 GHz (80 virtual cores), 726 GB DDR4 DRAM @ 2666 MT/s, 40 Gb Ethernet, 2 P100 GPUs, and running CentOS Linux 7.9.

We use PyTorch Geometric 1.7.2 [5] as our graph embedding implementation framework, PyTorch 1.9.0 [17] as the deep learning backend framework, and PyTorch *DistributedDataParallel* for distributed training with NCCL [16] as the backend for collective communication operations for GPUs and *AllReduce* for gradient sharing.

3.2 Datasets

We experimented with two public benchmark datasets used for link prediction in KGs. The dataset statistics are summarized in Table 1. **FB15k-237** [2] is a subset of FreeBase [6] which contains facts extracted from Wikipedia. It consists of named entities and edges which determine the type of relation between entities. The dataset is a benchmark for evaluating KG embedding methods for link prediction. **ogbl-citation2** [9] is a citation graph extracted from Microsoft Academic Graph (MAG) [25]. Each vertex is associated with a 128-dimensional feature vector created using Word2Vec[15] on paper title and abstract.

Table 1. Dataset statistics.

Dataset	FB15k-237	ogbl-citation2
# Entities	14,541	2,927,963
# Relations	237	1
# Features	-	128
# Train edges	272,115	30,387,995
# Valid edges	17,535	86,596
# Test edges	20,466	86,596

We use vertex-cut based partitioning (KaHIP [19, 20]) to obtain disjoint subsets of training edges followed by neighborhood expansion to include all n-hop neighbors.

3.3 Hyperparameters

We used a two layer RGCN [21] model as a KG embedding method for link prediction and trained it in a distributed setting. For FB15k-237 dataset, we used the same hyperparameters as described in [21] except the embedding size. We found out that an embedding size of 75 dimensions produces comparable results with the original setting. As the dataset is small, we trained it using full edge batch. For ogbl-citation2, we chose an embedding size of 32, learning rate 0.01, dropout 0.2, 1 negative sample per positive sample, and basis decomposition with two basis functions for regularization. For fairness of comparison, we used these hyperparameters in all training scenarios including non-distributed training.

3.4 Distributed Training Results

In this section, we present the experimental results of our proposed approach. We compare our proposed distributed training with a non-distributed training setup (1 Trainer). The non-distributed training process trains on the full graph data.

3.4.1 Accuracy. We compare the performance of our distributed training approach with the non-distributed training setting. In case of distributed training, the number of trainers varies from 2 to 8 trainers with each compute node running two trainers. Since, FB15k-237 is a small dataset, we performed full batch training. In case of ogbl-citation2, we performed mini-batch training, and the mini-batch size is approximately 118k. For FB15k-237 dataset, we follow the filtered settings for evaluations. Since the number of edges for dataset ogbl-citation2 is very large, the dataset has provided 1000 candidate negative *target* vertices for each test and validation edge for evaluation. We trained RGCN models on the two datasets, selected the best models and report the results on the test data. Ogbl-citation2 reached

Table 2. MRR, Hits@1 and epoch time/speedup of RGCN non-distributed and distributed training on 2 datasets.

#Trainers	FB15k-237			ogbl-citation2		
	MRR	Hits@1	Ep. Time(s)/speedup	MRR	Hits@1	Ep. Time(m)/speedup
1	0.22	0.138	5.09/-	0.620	0.494	112/-
2	0.22	0.136	4.03/1.25x	0.621	0.492	44/2.54
4	0.21	0.130	3.62/1.40x	0.620	0.493	16/7x
8	0.21	0.124	3.54/1.43x	0.617	0.494	7/16x

maximum accuracy within 100 epochs. As shown in Table 2, the results (MRR and Hits@k) on benchmark dataset FB15k-237 shows that our distributed training approach produces achieves comparable MRR and Hits@1 scores to the non-distributed setting. The results are also comparable to the numbers reported in the original RGCN paper. For the larger dataset, ogbl-citation2, we observe a similar trend, i.e our distributed training approach achieves comparable results to non-distributed training at a speedup of 16×. Furthermore, the results also verify that our constraint-based negative sampling strategy is effective in training as no deterioration of the scoring metrics is observed.

3.4.2 Scalability. We used the same settings as described in Section 3.4.1 for evaluating the scalability of our approach. We did not apply any sampling strategy (vertex drop or edge drop) during training. As shown in Table 2, the speed up for FB15k-237 dataset is lower than linear. The size of the partitions after neighborhood expansion is approximately equal to the original dataset with a replication factor for 8 partitions of around 7, which leads to a lot of redundant computations. For ogbl-citation2, we achieved a speedup of approximately 16x with 8 trainers compared to 1 trainer. The replication factor for 8 partitions after neighborhood expansion is low, which means less redundant computation.

We performed an in-depth analysis of the running times of the major computational components described in lines 4-10 of Algorithm 1 (*getComputeGraph*, *GNNmodel*, *loss+backward+step*) in order to quantify the contributions of these components to the overall speedup. Figure 5(a,b) shows the average epoch time and average running time of different components in a batch. The component, *getComputeGraph* is a very compute intensive operation as it depends on the partition size. The function returns a computational graph for an edge mini-batch. It is an essential function as it enables us to train on large graphs. The running time of this operation decreases as we increase the number of trainers from 1 to 8, because the size of the partitions decreases. In the case of 8 partitions for ogbl-citation2, the size of each partition decreases by one-third with respect to the full graph. The *GNNmodel* produces the embeddings of the vertices in an edge mini-batch. Its running time in case of multiple trainers is slightly higher than for 1 trainer, because we run 2 trainers per machine which share the same resources. The running time of the third block of operations (*loss+backward+step*) increases as we increase the number of trainers, because of the increase in communication cost for gradient sharing. The overall impact of these components on training time varies because the number of batches (forward pass and backward pass) per epoch decreases from 256 to 32 for 1 trainer and 8 trainers respectively.

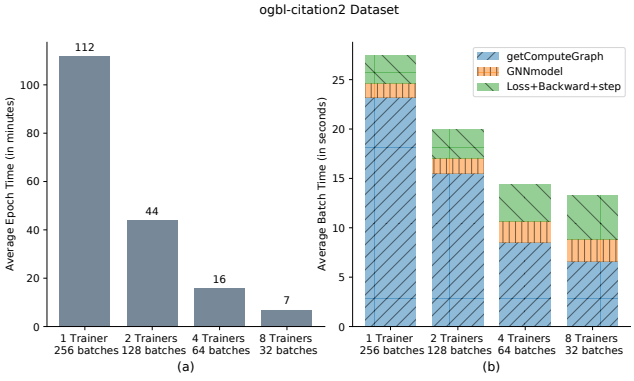

Figure 5. (a) Average running time per epoch; (b) average running time of components in batch for ogbl-citation2 dataset

4 Conclusion

We proposed various algorithmic approaches for distributed training of GNN-based knowledge graph embedding models. Our approach is agnostic to the used knowledge graph embedding model. We used a vertex cut partitioning approach along with neighborhood expansion method to make the partitions self-sufficient such that no data is transferred across partitions during training. We introduced edge mini-batch training for large partitions that enables us to train on large partitions with limited system memory. Moreover, we applied constraint-based negative sampling to exploit the local partitions to generate the negative samples for training. Our experimental evaluation shows a super linear speedup on a cluster of machines without sacrificing model accuracy, and our approach converges faster than non-distributing training.

References

[1] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale Simple Question Answering with Memory Networks. arXiv:1506.02075

[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-Relational Data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems* (Lake Tahoe, Nevada) *(NIPS'13)*. Red Hook, NY, USA, 2787–2795.

[3] Liwei Cai and William Yang Wang. 2017. KBGAN: Adversarial Learning for Knowledge Graph Embeddings. *CoRR* abs/1711.04071 (2017). arXiv:1711.04071 http://arxiv.org/abs/1711.04071

[4] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. 2015. One Trillion Edges: Graph Processing at Facebook-Scale. *Proc. VLDB Endow.* 8, 12 (2015), 1804–1815. https://doi.org/10.14778/2824032.2824077

[5] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[6] Google. 2013. Freebase Data Dumps. https://developers.google.com/freebase.

[7] Masatoshi Hanai, Toyotaro Suzumura, Wen Jun Tan, Elvis Liu, Georgios Theodoropoulos, and Wentong Cai. 2019. Distributed Edge Partitioning for Trillion-Edge Graphs. *Proc. VLDB Endow.* 12, 13 (Sept. 2019), 2379–2392.

[8] Yanchao Hao, Yuanzhe Zhang, Kang Liu, Shizhu He, Zhanyi Liu, Hua Wu, and Jun Zhao. 2017. An End-to-End Model for Question Answering over Knowledge Base with Cross-Attention Combining Global Knowledge. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 221–231. https://doi.org/10.18653/v1/P17-1021

[9] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* (2020).

[10] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (Dec. 1998).

[11] Bhushan Kotnis and Vivi Nastase. 2017. Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs. *CoRR* abs/1708.06816 (2017).

[12] Adam Lerer, Ledell Wu, Jiajun Shen, Timothée Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-BigGraph: A Large Scale Graph Embedding System. In *Proc. of Machine Learning and Systems 2019, MLSys 2019*.

[13] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (Austin, Texas) *(AAAI'15)*. AAAI Press.

[14] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2019. NeuGraph: Parallel Deep Neural Network Computation on Large Graphs. In *2019 USENIX Annual Technical Conference*.

[15] Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *CoRR* abs/1310.4546 (2013). arXiv:1310.4546 http://arxiv.org/abs/1310.4546

[16] NVIDIA. 2017. Optimized primitives for inter-GPU communication. https://github.com/NVIDIA/nccl.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[18] Xiao Qin, Nasrullah Sheikh, Berthold Reinwald, and Lingfei Wu. 2021. Relation-aware Graph Attention Model with Adaptive Self-adversarial Training. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 11 (May 2021), 9368–9376.

[19] Peter Sanders and Christian Schulz. 2013. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13) (LNCS, Vol. 7933)*. Springer, 164–175.

[20] Sebastian Schlag, Christian Schulz, Daniel Seemaier, and Darren Strash. 2018. Scalable Edge Partitioning. arXiv:1808.06411 [cs.DS]

[21] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*. Springer.

[22] Nasrullah Sheikh, Xiao Qin, Berthold Reinwald, Christoph Miksovic, Thomas Gschwind, and Paolo Scotton. 2021. Knowledge Graph Embedding using Graph Convolutional Networks with Relation-Aware Attention. *CoRR* abs/2102.07200 (2021). arXiv:2102.07200 https://arxiv.org/abs/2102.07200

[23] Zequn Sun, Qingheng Zhang, Wei Hu, Chengming Wang, Muhao Chen, Farahnaz Akrami, and Chengkai Li. 2020. A Benchmarking Study of Embedding-Based Entity Alignment for Knowledge Graphs. *Proc. VLDB Endow.* 13, 12 (July 2020), 2326–2340. https://doi.org/10.14778/3407790.3407828

[24] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*.

[25] Kuansan Wang, Iris Shen, Charles Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft Academic Graph: when experts are not enough. *Quantitative Science Studies* 1, 1 (February 2020), 396–413. https://www.microsoft.com/en-us/research/publication/microsoft-academic-graph-when-experts-are-not-enough/

[26] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

[27] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. 2018. Cross-lingual Knowledge Graph Alignment via Graph Convolutional Networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 349–357.

[28] Konstantinos Xirogiannopoulos, Udayan Khurana, and Amol Deshpande. 2015. Graphgen: Exploring interesting graphs in relational data. *Proceedings of the VLDB Endowment* 8, 12 (2015), 2032–2035.

[29] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *3rd Int. Conference on Learning Representations, ICLR San Diego, CA, USA*.

[30] Rui Ye, Xin Li, Yujie Fang, Hongyu Zang, and Mingzhong Wang. 2019. A Vectorized Relational Graph Convolutional Network for Multi-Relational Network Alignment. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (Macao, China) *(IJCAI'19)*. AAAI Press, 4135–4141.

[31] Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. 2017. Graph Edge Partitioning via Neighborhood Heuristic. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 10 pages. https://doi.org/10.1145/3097983.3098033

[32] Ningyu Zhang, Shumin Deng, Zhanlin Sun, Guanying Wang, Xi Chen, Wei Zhang, and Huajun Chen. 2019. Long-tail Relation Extraction via Knowledge Graph Embeddings and Graph Convolution Networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 3016–3025.

[33] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs. In *10th IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms, IA3 2020, Atlanta, GA, USA, November 11, 2020*. IEEE, 36–44. https://doi.org/10.1109/IA351965.2020.00011

[34] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. DGL-KE: Training Knowledge Graph Embeddings at Scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 739–748.

[35] Zhaocheng Zhu, Shizhen Xu, Meng Qu, and Jian Tang. 2019. GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding. In *The World Wide Web Conference*. ACM, 2494–2504.