CPa-WAC: Constellation Partitioning-based Scalable Weighted Aggregation Composition for Knowledge Graph Embedding

Sudipta Modak[1,2] , **Aakarsh Malhotra**[2] , **Sarthak Malik**[2] , **Anil Surisetty**[2] , **Esam Abdel-Raheem**[1]
[1]Department of Electrical and Computer Engineering, University of Windsor, ON, Canada
[2]AI Garage, Mastercard, Gurugram, Haryana, India
{modak, eraheem}@uwindsor.ca, {aakarsh.malhotra, sarthak.malik, anil.surisetty}@mastercard.com

Abstract

Scalability and training time are crucial for any graph neural network model processing a knowledge graph ($\mathcal{KG}$). While partitioning knowledge graphs helps reduce the training time, the prediction accuracy reduces significantly compared to training the model on the whole graph. In this paper, we propose *CPa-WAC*: a lightweight architecture that incorporates graph convolutional networks and modularity maximization-based constellation partitioning to harness the power of local graph topology. The proposed *CPa-WAC* method reduces the training time and memory cost of knowledge graph embedding, making the learning model scalable. The results from our experiments on standard databases, such as Wordnet and Freebase, show that by achieving meaningful partitioning, any knowledge graph can be broken down into subgraphs and processed separately to learn embeddings. Furthermore, these learned embeddings can be used for knowledge graph completion, retaining similar performance to training a GCN on the whole $\mathcal{KG}$, while speeding up the training process by upto five times. Additionally, the proposed *CPa-WAC* method outperforms several other state-of-the-art $\mathcal{KG}$ in terms of prediction accuracy.

1 Introduction

Knowledge graph ($\mathcal{KG}$) has gained immense popularity in recent years. A $\mathcal{KG}$ is a diverse multigraph consisting of more than one type of directed relation [Zamini *et al.*, 2022] between node entities. Each $\mathcal{KG}$ contains a collection of facts organized as a graph to represent a group of linked entities and their semantic descriptions [Zamini *et al.*, 2022]. First proposed in 2012 by Google, these semantic networks are widely used today, from drug interaction prediction in chemistry to fraud detection in financial transactions to entity alignment [Surisetty *et al.*, 2022; Chaurasiya *et al.*, 2022]. To date, many $\mathcal{KG}$s have been created, such as Freebase [Bollacker *et al.*, 2008] and Wordnet [Miller, 1995], which illustrate real-life relations between entities. Knowledge graphs are almost always incomplete [Ye *et al.*, 2022]. Therefore, to complete such graphs, it is essential to embed them and use these embeddings to predict relations between entities. Knowledge graph embedding (KGE) is considered a foundation for several prediction tasks using $\mathcal{KG}$ [Ye *et al.*, 2022].

Many methods utilize neural networks to compute embeddings of nodes and relations and use them for inference. Similarly, some methods employ deep learning to learn recursive logical rules [Meilicke *et al.*, 2019; Cheng *et al.*, 2022] for reasoning over $\mathcal{KG}$. Methods such as graph neural networks (GNN) [Wu *et al.*, 2020; Surisetty *et al.*, 2023] are neural network-based frameworks that have recently been integrated into the $\mathcal{KG}$ domain and are primarily used to learn embeddings of nodes and relations. Relational graph convolutional networks (RGCN) [Schlichtkrull *et al.*, 2018] was the first GCN-based method introduced in the field. Since then, more sophisticated approaches have been developed. Graph convolutional and attention networks (GAT) [Veličković *et al.*, 2018] have also been quite successful in achieving high accuracy of prediction, superseding convolutional networks [Dettmers *et al.*, 2018]. However, the information-rich embeddings from these algorithms come at a significant cost of computational and memory usage.

Several state-of-the-art algorithms such as Comp-GCN [Vashishth *et al.*, 2019], RAGAT [Liu *et al.*, 2021], and SE-GNN [Li *et al.*, 2022a] use high amounts of memory (GPU) and require immense amounts of training time. With a large number of nodes and edges, the time complexity to process these graphs can be considered quadratic [Wu *et al.*, 2020] due to millions of trainable parameters. As a result, several of these algorithms can only be trained with small batch sizes. Partitioning $\mathcal{KG}$ into subgraphs and processing these subgraphs can be depicted as a solution to this problem [Puja *et al.*, 2013]. Partitioning reduces the number of computations in both forward and backpropagation. At the same time, it enables the training process to be carried out parallelly. While a few methods employ partitioning and parallel training, such as the ones presented in [Kochsiek and Gemulla, 2021; Sheikh *et al.*, 2022; Bai, 2023], it is evident that dedicated partitioning approaches are required that can partition $\mathcal{KG}$s without the utilization of node or edge features.

This research presents a novel composition-based KGE method using graph neural networks that utilizes the power of $\mathcal{KG}$ partitioning to make the training process fast and efficient. We call the proposed algorithm constellation partitioning-based weighted aggregation composition (*CPa-*

WAC). We harness the power of modularity maximization [Newman, 2006] through Louvain clustering to generate partitions from $\mathcal{KG}$. By preserving the original graph topology, the proposed method increases link prediction performance compared to other partitioning-based state-of-the-art methods while decreasing the train time. Furthermore, we compare the Louvain algorithm to the Leiden algorithm [Traag *et al.*, 2019] and empirically verify the effects of utilizing both algorithms on $\mathcal{KG}$ partitioning.

The main contributions of this paper are as follows: i) A novel $\mathcal{KG}$ partitioning algorithm (*CPa*) that utilizes fast Louvain clustering [Blondel *et al.*, 2008] to partition the $\mathcal{KG}$ into several topological clusters. During partitioning, we aim to minimize the number of lost links between clusters; ii) An improved compositional-GCN algorithm, *WAC*. The composition operation is further coupled with the multiplication operation that is presented in [Kazemi and Poole, 2018] and a 1D convolutional network that takes advantage of feature, entity, and relation-specific weights to learn effective embeddings; iii) A global decoder framework that can use node and relationship embeddings from different clusters to achieve a global-level inference, and lastly (iv) exhaustive comparison on four distinct datasets, namely, WN18, WN18RR [Miller, 1995], FB15K, and FB15K-237 [Bollacker *et al.*, 2008] and several state-of-the-art methods in the field, along with a detailed analysis of each comparison is presented in the paper.

2 Related Work

2.1 Knowledge Graph Embedding

Several methods have been used to train a $\mathcal{KG}$ and obtain embeddings. Models such as the variants of Trans: TransE [Bordes *et al.*, 2013], TransH [Wang *et al.*, 2014], TransR [Lin *et al.*, 2015], TransD [Ji *et al.*, 2015], and TransG [Wang *et al.*, 2017] were pivotal for KGE towards link prediction, node classification, and reasoning tasks. However, more complex semantic methods aimed to capture semantic relationships, such as Conv2D [Dettmers *et al.*, 2018], RESCAL [Bordes *et al.*, 2014], ComplEX [Zhou *et al.*, 2017], TuckER [Balazevic *et al.*, 2019], HAKE [Zhang *et al.*, 2020], and SimplE [Kazemi and Poole, 2018]. While these methods are good at capturing semantic relationships, they require a high dimensionality in the embedding space [Chen *et al.*, 2020] to embed entities and relationships for more accurate predictions.

Recent architectures such as GCN [Schlichtkrull *et al.*, 2018] and graph attention networks (GAT) [Veličković *et al.*, 2018] have been utilized in studies such as [Vashishth *et al.*, 2019; Liu *et al.*, 2021; Li *et al.*, 2022a; Li *et al.*, 2022b]. If pruned properly, these architectures produce better results than their non-GCN-based counterparts. Nevertheless, GCN and GAT-integrated models have high trainable parameters and take enormous time to train on $\mathcal{KG}$ datasets. Even though there are libraries available, such as Pytorch-Biggraph [Lerer *et al.*, 2019] and DGL-KE [Zheng *et al.*, 2020], these libraries do not address the scalability of GNN-based knowledge graph embedding algorithms.

2.2 Reducing Training Time of Knowledge Graphs

One way to reduce training time lies in the task of $\mathcal{KG}$ augmentation. $\mathcal{KG}$ augmentation can be used to learn embeddings in a lower dimensional vector space. As demonstrated by [Wang *et al.*, 2022a], $\mathcal{KG}$ augmentation, when used in sequence with a Trans or semantic series embedding model, can reduce the processing time and increase the overall performance of these models. Methods such as GreenKGC [Wang *et al.*, 2022b] and DGL-KE [Zheng *et al.*, 2020] have addressed the reduction of embedding dimensionality and the training time using feature pruning, partitioning, parallel training, and multi-GPU training.

Partitioning is critical to scalable GNN-based $\mathcal{KG}$ embedding generation. Partitioning can be used to split a $\mathcal{KG}$ into multiple subgraphs without destroying the original topology of the graphs. This statement is backed by the analysis of Jain *et al.* in their work presented in [Jain *et al.*, 2021], which showcased that the semantic representations of embeddings are not foolproof over the entire graph. Instead, semantic features are only contained locally, implying that partitioning will not distort the overall graph structure. Ontology-based $\mathcal{KG}$ partitioning has been employed by Bai in his work in [Bai, 2023], which breaks the $\mathcal{KG}$ into multi-partitions. The method shows high performance on partitioned datasets, compared to the performance on the full dataset. Similarly, METIS [Karypis and Kumar, 1998] and k-means clustering have been utilized in [Wang *et al.*, 2022b] and [Zheng *et al.*, 2020], respectively, followed by Trans series algorithms to partition $\mathcal{KG}$ into disjoint clusters to reduce the training time. Apart from traditional k-means and METIS partitioning, some novel $\mathcal{KG}$ partitioning methods are presented in the works [Sheikh *et al.*, 2022] and [Priyadarshi and Kochut, 2021]. The work in [Sheikh *et al.*, 2022] presents an edge-cut partitioning method that can aggregate information from only essential nodes from connected subgraphs after partitioning. Similarly, the work presented in [Priyadarshi and Kochut, 2021] has proposed a workload-aware knowledge graph partitioning that analyses training triples and the connected $\mathcal{KG}$ to form partitions.

Despite several studies on $\mathcal{KG}$ partitioning-based embedding models, properly partitioning a $\mathcal{KG}$ with the least cross partition edges remains challenging. Furthermore, a framework is required to merge these individual embeddings into one complete graph structure. Considering these methods in the field, a dedicated $\mathcal{KG}$ partitioning architecture is required that ensures the least cross-cluster relations. Furthermore, developing a lightweight GCN-based architecture is essential, which can be coupled with partitioning to learn effective embeddings from $\mathcal{KG}$ faster. In this paper, we address this problem by proposing a novel $\mathcal{KG}$ partitioning method and a composition-based aggregation method to achieve faster training while keeping the prediction accuracy high.

3 Materials and Methods

The methodology is divided into three stages: Partition generation (3.2), Weighted Aggregation Composition Convolution (3.3), and the partitioning-based decoder framework for link prediction, (3.4). The proposed *CPa-WAC* algorithm is summarized in Figure 1. The code is made available for the research community[1], and summarized in Algorithm 1.

3.1 Problem Definition and Notations

A knowledge graph is a semantic network, represented as $\mathcal{KG}$ = $\{E, R, T\}$. Here E, R, and T represent entities, relations, and triples, respectively. The triples in a $\mathcal{KG}$ can be regarded as T = $\{\varrho$, μ, $\xi\}$, where ϱ, μ, and ξ symbolizes a head, relation, and tail, respectively. The relations are directed and represent a link between two entities. This study aims to empirically verify that partitioning can be used to speed up KGE without destroying the structure of the $\mathcal{KG}$ and jumbling the inference logic.

3.2 Constellation Partitioning: *CPa*

As the first step towards speed-up, the proposed method employs Louvain clustering (LC) [Blondel *et al.*, 2008] as its base. The essence of LC lies in community detection/subgraph generation, which partitions a given homogeneous graph based on the density of its edges. This method exclusively suits graphs without node and edge attributes, making it a perfect fit for $\mathcal{KG}$ partitioning. The proposed constellation partitioning (*CPa*) algorithm utilizes the LC algorithm on $\mathcal{KG}$ to partition it into several topological structures. The process starts by creating a symmetric adjacency matrix (**A**). If a link exists between two nodes in a graph, then a 1 is added at positions in **A** representing the connection. This process is repeated for every type of relation for all nodes. This weighted adjacency matrix can be fed to the LC algorithm to obtain the initial number of clusters using modularity score maximization. The modularity score M_c for each cluster c is given as:

$$M_c = \frac{\alpha_{in}}{2\omega} - \left(\frac{\alpha_{all}}{2\omega}\right)^2, \quad (1)$$

Here, α_{in} is the summation of the total weights of all links contained only within cluster c. Similarly, α_{all} represents the summation of the weights of links that the nodes of cluster c have with each other and also with nodes from other clusters. The variable ω is the total weight of all links in the graph. Once the initial number of clusters are formed using the LC algorithm, using a hard threshold of δ, clusters having only a few entities are all merged into one cluster. This groups those few outlier entities in $\mathcal{KG}$ that have very few links and can distort the embedding process. Here, threshold δ is a $\mathcal{KG}$-specific hyperparameter.

Despite creating one merged cluster with a threshold δ, some clusters with a small number of entities may remain. This is due to the LC algorithm's limitation in partitioning heterogeneous directed graphs. Therefore, merging these small clusters with bigger and denser clusters is essential in avoiding loss of structural information. To do so, we utilize three levels of hierarchical merging using Φ=$\gamma \times \beta \times \delta$. Here, δ is the initial threshold from the previous step, γ is the incremental step, and β is also a hyperparameter to control the number of clusters created. This finally gives us the C number of clusters. The logic is that if the number of entities in a cluster is below a threshold Φ, it will be merged with a cluster

[1]https://github.com/ganzagun/CPa-WAC

Algorithm 1 Louvain Constellation Partitioning

```
1:  **Initialize**: **A**, **A'** as square matrices of zeroes with each
    dimension equal to the number of nodes, $N$
2:  **Initialize**: $\gamma$, $\beta$, $\delta$, $\sigma$
3:  **For** $i$ = 1 : length($T$)
4:      Add 1 to positions in $A_{(T_{(i,0)},T_{(i,2)})}$
5:      Add 1 to positions in $A_{(T_{(i,2)},T_{(i,0)})}$
6:      $A'_{(T_{(i,0)},T_{(i,2)})}$=1
7:      $A'_{(T_{(i,2)},T_{(i,0)})}$=1
8:  **End**
9:  Apply LC to **A**
10: Maximize modularity using Equation 1
11: Cluster outliers with threshold $\delta$
12: **For** $j$ = 1 : 3 (for three levels of hierarchical merging)
13:     Obtain $k$ number of clusters and store in vector **L** to
        map each entity to a cluster
14:     Create a list U with all entities present in each
        cluster
15:     **For** $m$ = 1 : $N$
16:         **B** = $A'_{(m)}$ × **L**
17:         The label at position $A'_{(m,m)}$ = $l$
18:         **For** $p$ = 1: $k$
19:             $x$ = Number of entities in **B** with label $p$
20:             Store $x$ in vector **F** at position $F_{(p)}$
21:         **End**
22:         Find argmax(**F**) whose label is not $l$
23:         Store $p$ in vector **M** at position $M_m$. This is the
            cluster that is adjacent to node $m$
24:     **End**
25:     **For** $q$ = 1 : $k$
26:         Identify positions in vector **L** with label $q$
27:         Take the corresponding positions from vector **M**
            and store them in vector **D**
28:         Based on Majority voting of nodes in vector **D**
            select the nearest cluster, $y$
29:         $\Phi$= $\gamma \times \beta \times \delta$
30:         **If** length(U($y$)) $< \sigma$ and length(U($q$)) $< \Phi$
31:             Merge the two clusters by applying labels from
                the bigger cluster $y$, to the smaller cluster $q$
32:         **End**
33:     **End**
34:     $\gamma$= 2 * $\gamma$ (incremental step)
35: **End**
```

with a higher number of entities with the most links. We call this the nearest linked neighbor (NLN). The three incremental thresholds applied to the clusters ensure steady growth of entities in highly dense clusters. To avoid entity explosion, a second threshold of σ is used to cap the maximum number of entities in a cluster. If the number of entities in a cluster $> \sigma$, then a smaller cluster will not be merged with it even if this cluster is its NLN. Hence, for clusters containing a higher number of entities, smaller clusters that are designated as NLN will be merged iteratively with them. Thus, a relatively similar number of entities is obtained for each cluster while the overall modularity of the graph remains high.

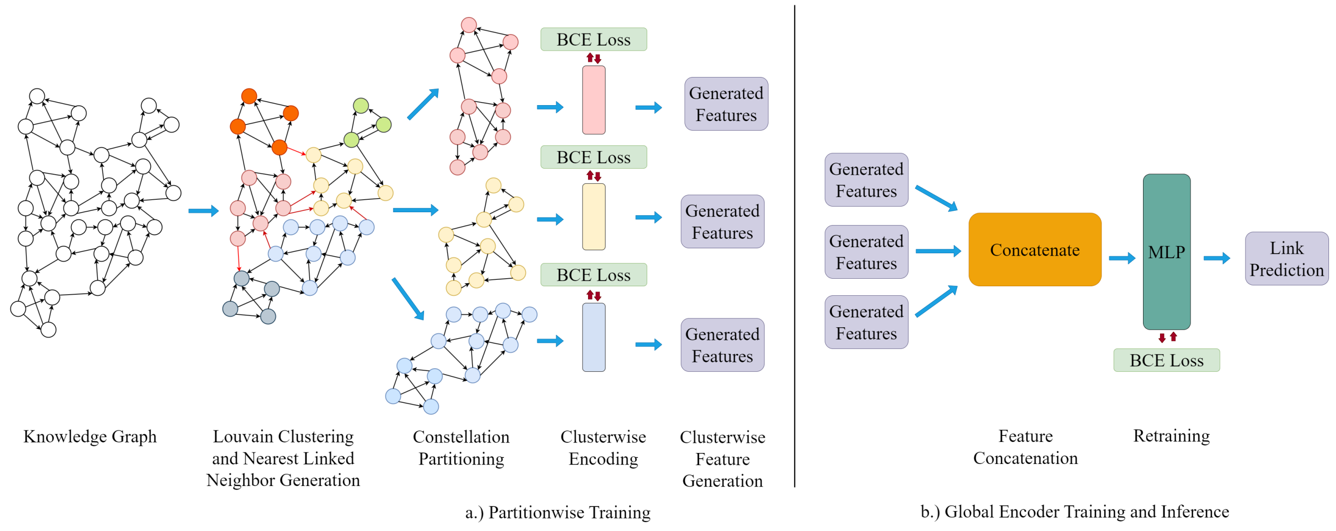
BCE Loss
Generated Features
Concatenate
MLP
Link Prediction
Knowledge Graph
Louvain Clustering and Nearest Linked Neighbor Generation
Constellation Partitioning
Clusterwise Encoding
Clusterwise Feature Generation
Feature Concatenation
Retraining
a.) Partitionwise Training
b.) Global Encoder Training and Inference

Figure 1: Block diagram of the proposed method a.) Using the LCP to divide the training triples for different clusters and train the partitions separately. b.) Concatenation of node and relation features along with retraining the Global encoder for link prediction.

3.3 Weighted Aggregation Composition *(WAC)* Convolution

In this section, we present an improved version of compositional message-passing [Vashishth *et al.*, 2019], namely Weighted Aggregation Composition (*WAC*) Convolution. We harness the power of graph attention layers [Liu *et al.*, 2021] to obtain the embeddings of entities and relations by passing through two different composition functions for message aggregation. These functions are denoted by,

$$h_u = e_u^g \times w_u^g \times r_u^g, \tag{2}$$

$$h_v = \Sigma_{b=1}^{z} \Sigma_{v=1}^{f} (e_v^g \times w_v^g \times r_b^g + e_v^g \times w_b^g), \tag{3}$$

where e_u is the embedding of the entity number u and r is the relation embeddings. The variables, w_u, w_v, and w_b, denote the learnable weight vectors needed to transform the entities and relationships. Here, h_u and h_v represent the composition operations needed to transform a particular entity, once updating itself using its entity weight and self-loop (Eq. 2) and a second time by the weights and messages from its neighbors (Eq. 3). The term g represents the current state of embeddings, v represents the neighbor number of the u^{th} entity, b represents the relation type number out of z number of relations that connects the u^{th} entity to its neighbors, and f represents the total number of neighbors that the u^{th} entity is connected to, and, '$\times$' is element-wise feature multiplication.

Once h_v and h_u are calculated, an activation function is used for each level of message aggregation. The Gaussian error linear unit (GELU) activation function ψ is chosen to update the message-passing function. Next, the summation of neighborhood messages (h_v) is passed through an attention layer after normalizing the messages with the degree matrix $\mathbf{G}$. The attention layer, Γ, adds specific attention to all features for each entity. Finally, we take a weighted sum of h_v and h_u to produce the embeddings e_u. The entire process can be summarized using the following equation,

$$e_u^{g+1} = \zeta \times \psi(h_u) + \theta \times \psi(\Gamma(G_v^{-0.5} \times h_v \times G_v^{-0.5})), \tag{4}$$

where $g+1$ represents the next stage of the entity embeddings, and ζ and θ are scalar weights and are separate for each level of aggregation. The relation embeddings are updated using,

$$r_b^{g+1} = r_b^g \times v_b^g, \tag{5}$$

where v_b represents a separate learnable weight vector from w_b and can be updated using backpropagation to emphasize more dominant relations in the $\mathcal{KG}$.

Finally, a 1D convolutional neural network (1D-CNN) decodes the embedding. It uses a multiplication operation of entities and relation embeddings, similar to the work in [Kazemi and Poole, 2018]. This module consists of several filters and a dense layer, which is trained iteratively to produce meaningful features from obtained embeddings. Batch normalization is utilized twice, once after the convolutional process and the second after passing through the dense layer. The binary cross-entropy (BCE) loss function trains models on each cluster of entities and relationships separately.

3.4 Global Decoder (GD) Framework & Inference

Once the embeddings for entities and relations are obtained, the method moves to its last stage where a separate framework is created and trained for inference. This module is named the global decoder (GD) framework entire training data is fed to this network a second time to train the weights of the MLP

This framework concatenates the feature vectors for all nodes and relations. Let an entity embedding e_u^c for u^{th} entity from a cluster c be of dimension $1 \times s$. Considering C generated clusters, embedding e_u^c is upscaled to a dimension $1 \times Cs$. In upscaled e_u^c, a portion of values is the same embedding, while the remaining $1 \times (C-1)s$ values are zeros. These features are then fed into an MLP after multiplying the embeddings with trainable weight matrices, $\mathbf{W}_e$ and $\mathbf{W}_r$. Here, $\mathbf{W}_e$ and $\mathbf{W}_r$ are the weight matrix that projects the upscaled entity and relation embeddings to lower dimensions, respectively. These weights are then trained along with the weights for the MLP in an end-to-end fashion utilizing a multiclass BCE loss. The final model and the embeddings of nodes and relations are then used for inference. For testing, the nodes and relations of the test set are fed to the model to predict the tail. Similarly, the reverse is done for predicting the head.

4 Experimental Setting

4.1 Implementation and Database Details

All experiments have been conducted on an I7-13700, 2.1 GHz system with 32 GB RAM and NVIDIA RTX A2000 12 GB GPU. The AdamW optimizer is utilized to train the weights of the proposed architecture for a total of 400 epochs for all partition-based experimentation. Consequently, for comparing the *WAC* convolution to other architectures, we show results for a model trained on 400 and 1500 epochs (as some papers in literature use 1500 epochs as their setting). Furthermore, all state-of-the-art models have been implemented using the same hyperparameter settings as the proposed architecture. This setting includes batch number, regularization rate, weight decay, learning rate, and the same embedding dimensions for each model.

As summarized in Table 1, we use the most widely used Wordnet [Miller, 1995] and Freebase [Bollacker *et al.*, 2008] for our experiments. Two versions of each dataset are used, namely WN18 and WN18RR for Wordnet and FB15K and FB15K-237 for Freebase. These four datasets capture different denseness and scales of the graph. Particularly, FB15K and FB15K-237 are large-scale $\mathcal{KG}$, making them vital for research on scalable $\mathcal{KG}$.

Dataset	E	R	Training	Valid	Test
FB15K	14,951	1,345	483,142	50,000	59,071
FB15K-237	14,541	237	272,115	17,535	20,466
WN18	40,943	18	141,442	5,000	5,000
WN18RR	40,943	11	86,835	3,034	3,134

Table 1: Description of Knowledge Graph Datasets.

4.2 Evaluation Metrics

We evaluate the proposed method using popular KGE evaluation metrics of mean reciprocal rank (MRR) and $Hits@K$. Furthermore, partitioning is evaluated using a novel proposed evaluation metric, that is, **t**ail and **h**ead in the same **p**artition (THP). The metrics are defined as follows:

$$MRR = \frac{1}{Q}\Sigma_{d=1}^{Q}\frac{1}{rank_d}, \tag{6}$$

$$Hits@K = \frac{o_+}{S_{all}}, \tag{7}$$

$$THP = \frac{\Sigma_{n=1}^{C}S_n}{S_{all}} \times 100, \tag{8}$$

where Q is the number of reciprocal ranks, d is the element number, and $rank$ is the position of the highest-ranked answer $(1, 2, 3, ..., X)$ for X answers returned in a query. Here, o_+ is the number of positives occurring in the top-X positions, and S_{all} is the total number of triples in the dataset. For C number of obtained partitions, THP can be defined as the number of triples S_n with the head and tail within the same partition compared to the total number of triples, S_{all}.

5 Results and Analysis

This section talks about results and comparisons from proposed *CPa-WAC* and other state-of-the-art methods in the field of $\mathcal{KG}$ partitioning and optimization.

5.1 Partition Analysis

Using THP as an evaluation metric for partitioning, we evaluate the proposed partitioning method *CPa* over four $\mathcal{KG}$ datasets in Table 2. Over varying cluster counts, we observe that over 70% of the training triples have their head and tail within the same partition. This even true at as high as eight separate clusters. For the test and validation sets, except for dataset FB15k-237, all datasets show a high THP over 70%. Despite THP being inversely proportional to the number of clusters, the proposed *CPa* method can retain much of the original structure of the $\mathcal{KG}$ even after partitioning. This is particularly evident for dense $\mathcal{KG}$ such as FB15k and WN18.

5.2 Embedding Method Analysis

As the next step, we analyze the effectiveness of the proposed *WAC* Convolution method on link prediction and compare it to state-of-the-art methods that employ GNN in their architectures. The comparison is done in terms of the MRR and average time per epoch of training (T/epoch) in seconds. For this experiment, we consider the whole $\mathcal{KG}$ for each dataset with predefined training, valid, and test sets.

From Table 3, we observe that the proposed method shows slightly better performance in terms of MRR. Alongside marginally better performance, the reduction in time for each epoch is considerably high, giving *WAC* an edge over the rest in terms of faster training of embeddings. The proposed *WAC* not only improves the MRR for denser graphs such as FB15k and WN18 but also speeds up the training immensely for these graphs. For sparser graphs such as FB15k-237 and WN18RR, while the MRR does not show much improvement, the performance speedup is twice compared to SE-GNN [Li *et al.*, 2022a] and approximately 1.7 times compared to RAGAT [Liu *et al.*, 2021].

5.3 Partition Performance Analysis

We analyze the acceleration rate in comparison to the performance drop by applying the proposed method on several $\mathcal{KG}$. Figure 2 depicts the effects of *CPa* coupled with LC and *WAC* on MRR and $Hits@K$ for C number of clusters. In this experiment, we train a separate architecture to infer links using our global decoder framework. In this setting, the obtained features from partitions are used to retrain the dense layers as described in Section 3.4.

From the experiments, we can conclude that disjoint subgraphs created by the proposed *CPa-WAC* method retains the essence of the original structure. With a global decoder, *CPa-WAC* outperforms, with the observed MRR and $Hit@K$

Dataset	Split	Number of Partitions (C)						
		2	3	4	5	6	7	8
FB15k-237	Train	99.98%	85.28%	79.50%	N/A	73.30%	73.09%	72.50%
	Valid	99.82%	81.25%	72.72%	N/A	65.15%	64.86%	64.06%
	Test	99.91%	81.79%	73.08%	N/A	65.59%	65.49%	64.45%
FB15k	Train	99.89%	89.44%	87.87%	75.12%	74.14%	73.99%	73.54%
	Valid	99.88%	89.23%	87.63%	74.34%	73.4%	73.26%	72.52%
	Test	99.86%	89.24%	87.56%	74.15%	73.11%	72.98%	72.78%
WN18RR	Train	100%	93.97%	94.38%	N/A	92.17%	N/A	90.71%
	Valid	92.76%	79.64%	81.49%	N/A	76.00%	N/A	73.20%
	Test	92.56%	80.32%	80.52%	N/A	77.19%	N/A	73.60%
WN18	Train	100%	94.43%	92.79%	91.22%	N/A	90.79%	90.11%
	Valid	100%	88.46%	85.46%	81.82%	N/A	81.16%	79.40%
	Test	100%	89.62%	86.24%	83.34%	N/A	83.08%	81.12%

Table 2: Partition Analysis: Evaluation of the proposed *CPa* coupled with LC in terms of head and tail achieved in the same partition (THP). The method is evaluated on different numbers of partitions on each dataset using the training set to generate partitions. Here N/A indicates that the proposed method did not generate the specific number of clusters for the dataset.

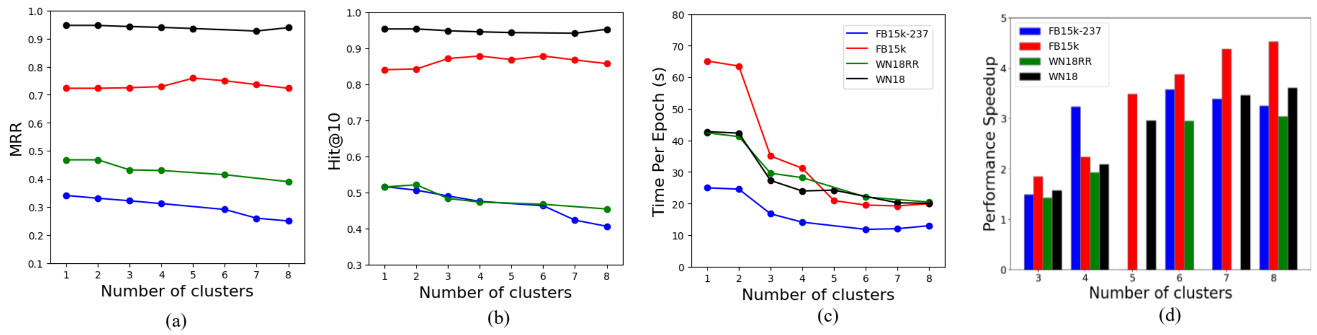

Figure 2: Performance of various datasets with different numbers of partitions produced using LC, *CPa*, *WAC* and GDF (a) *MRR* at different numbers of generated partitions (b) *Hits@K* at different numbers of generated partitions (c) Time per epoch without parallelization at different numbers of generated partitions (d) Performance Speedup evaluation with parallelization.

Dataset	Method	Comp-GCN	RAGAT	SE-GNN	*WAC*
FB15k-237	**MRR (400)**	0.338	0.349	0.364	0.341
	MRR (1500)	0.350	0.355	N/A	0.351
	T/epoch (s)	57.92	65.20	113.33	24.98
FB15k	**MRR (400)**	0.600	0.598	O/M	0.723
	MRR (1500)	0.621	0.637	O/M	0.742
	T/epoch (s)	95.98	168.23	N/A	65.16
WN18RR	**MRR (400)**	0.452	0.472	0.482	0.468
	MRR (1500)	0.464	0.478	N/A	0.481
	T/epoch (s)	47.20	70.76	98.26	42.39
WN18	**MRR (400)**	0.938	0.941	0.940	0.947
	MRR (1500)	0.940	0.944	0.942	0.952
	T/epoch (s)	50.41	62.60	105.86	42.78

Table 3: Comparison of the proposed embedding method to state-of-the-art methods for 400 and 1500 epochs. The models are compared in terms of MRR and time required per epoch in seconds. Here O/M represents out-of-memory for a certain set of hyperparameters.

higher than the whole $\mathcal{KG}$ (for 4 and 5 different partitions). From Figure 2, we see that despite a slight reduction in MRR and $Hits@K$ with increasing partitions, the dip is small up to 6 partitions. Note that setting C=1 translates to processing the whole $\mathcal{KG}$ together. In most cases, the acceleration increases with the increase in the number of partitions. We further employ parallelization of partitions on a single GPU by invoking subprocesses. As the number of entities and relationships is far less for each partition compared to the original $\mathcal{KG}$, a single GPU can process these subgraphs separately. The system can be used to train models on eight separate partitions simultaneously with the same memory cost compared to processing the entire graph using *WAC*. Figure 2(d) represents the speedup obtained during the training process. This speedup is calculated as the ratio of time to processes C number of partitions parallelly against the processing time of the entire $\mathcal{KG}$ together using *WAC*. While the acceleration remains between 2 to 4 times the average training time for most datasets, FB15k shows the most acceleration going up to 4.5 times for 7 and 8 clusters. However, using a multi-GPU system can further accelerate the process. Figure 2(c) shows that for denser KG (FB15K and WN18), the acceleration is better than their sparser counterparts (FB15K-237 and WN18RR). This implies that density also affects speed-up.

Dataset	FB15k-237		FB15k		WN18RR	
Method	**MRR**	**Hit@10**	**MRR**	**Hit@10**	**MRR**	**Hit@10**
[Bai, 2023]	0.291	0.486	0.695	0.872	-	-
[Sheikh *et al.*, 2022]	0.220	0.138	-	-	-	-
[Wang *et al.*, 2022b]	**0.345**	**0.507**	-	-	0.411	0.491
LC + *CPa* + *WAC*	0.331	0.506	**0.759**	**0.878**	**0.468**	**0.521**

Table 4: Comparison with state-of-the-art methods on Partitioning-based Embedding generation and inference. (-) denotes unreported results for the respective dataset in the original paper.

We compare the best performance achieved by the proposed method for each dataset and compare them with state-of-the-art methods of $\mathcal{KG}$ partitioning. These results are illustrated in Table 4. The comparative methods are selected as all of these use partitioning to achieve faster training. Overall, the proposed method achieves the best performance and second best for the FB15k-237 dataset, as observed by improved MRR and *Hit@10*. Furthermore, on the FB15k-237 dataset, [Sheikh *et al.*, 2022] reports speed up using multi-GPU distributed training to be four times. In comparison, the proposed method shows a similar speed-up at 3.7 times with just a single GPU with a considerable improvement in MRR and Hit@10. Hence, we conclude that the proposed method shows an overall edge in performance compared to other methods in the field that combine partitioning and KGE.

6 Ablation Study

In this section, we conduct an ablation study to analyze the contribution of the *CPa* component in the proposed KGE method on three aspects. **(i)** We compare the test MRR and $Hit@K$ for an alternate setup: border entity addition (BEA). This method utilizes the addition of triples with a head or a tail in another partition to the training data of each partition. This will enable some entities to be present in more than one cluster. This experiment aims to study the effects of overlapping partitioning and deduce its effect on the performance of KGE; **(ii)** We conduct another study that utilizes separate models for inference. In this setting, we save each model separately for each partition and use the saved model (SM) for inference using a reasoning framework. Under this scenario, if the head of a test triple falls within a particular partition, then the tail will only be selected from the same partition. Thus, in this setup, the architecture can not predict any link that exists between partitions, and so the MRR for these triples is considered as 0. When inferred through separate embedding models, the drop in performance is significant; **(iii)** We rerun our experiments utilizing a different partitioning method, namely, Leiden partitioning (LeP) [Traag *et al.*, 2019]. Given both are modularity maximization methods, both can be used due to quicker runtime and find application in community detection in homogenous graphs. This algorithm utilizes modularity vertex partitioning that only considers links between nodes. Other methods like METIS and spectral clustering are computationally expensive.

Table 5 shows the outcome of different iterations on the performance of the architecture on four partitions. In this table, the LC+*CPa*+*WAC*+GD signifies the results from the proposed algorithm. As seen from the results, adding border entities degrades the performance of $\mathcal{KG}$ when relationship types are less (while increasing the training time slightly). However, for denser $\mathcal{KG}$ such as FB15k that have a high number of relationship types, the addition of border entities enhances the performance by a small margin at the expense of an increase in training time. Therefore, we conclude that for sparse graphs containing a lower number of relationship types, border entity addition has overall negative effects on the performance of the KGE algorithm. Contrarily, for dense $\mathcal{KG}$ with a high number of relationship types, positive effects on performance can be achieved at expense of higher time.

Dataset	FB15k		WN18	
Models	**MRR**	**T/epoch**	**MRR**	**T/epoch**
LC+ *CPa* + *WAC* + GD	0.729	31.18	0.940	23.98
LC+ *CPa* + *WAC* + SM	0.651	26.86	0.822	16.89
LC+ *CPa* + *WAC* + BEA + GD	0.734	52.49	0.934	28.55
LC+ *CPa* + *WAC* + BEA + SM	0.589	48.17	0.811	21.46
LeP + *CPa* + *WAC* + SM	0.591	24.16	0.815	16.80
LeP + *CPa* + *WAC* + GD	0.753	28.49	0.941	23.12

Table 5: Ablation study of different combinations of modules for four clusters generated on larger datasets (FB15k and WN18).

Furthermore, using Leiden clustering to initialize the partitions yields better performance for denser graphs such as FB15k while the performance remains comparatively the same for sparser graphs such as WN18. Similarly, LeP shows a reduction in the training time compared to the LC for both datasets. This highlights that the proposed constellation partitioning algorithm (CPa) works perfectly when coupled with any homogenous graph partitioning algorithm that only considers the modularity of graphs.

7 Conclusion

In this paper, we present a combination of a novel method of partitioning $\mathcal{KG}$ along with an improved GCN-based embedding architecture. Results show that the proposed architecture performs well with a significant reduction in time complexity for GCN-based embedding generation architectures. Additionally, a comparison of three state-of-the-art methods with the proposed method has also been provided in this study, highlighting their embedding performance on KGE. The results have conclusively pointed toward the superior performance of the proposed method, both at the partitioning and the embedding levels. For future studies, we intend to develop a sophisticated reasoning algorithm that would replace the global merging algorithm to reduce the retraining time. **From ethical standpoint**, we recommend selecting train time vs. performance trade-off based on the metrics, properties of the graph, and criticality of the application.

Acknowledgments

This work was supported by Mitacs through the Mitacs Accelerate Program.

References

[Bai, 2023] Yuhe Bai. A semantic partitioning method for large-scale training of knowledge graph embeddings. In *ACM Web Conference*, pages 573–577, 2023.

[Balazevic *et al.*, 2019] Ivana Balazevic, Carl Allen, and Timothy Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *Conference on Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, 2019.

[Blondel *et al.*, 2008] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[Bollacker *et al.*, 2008] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[Bordes *et al.*, 2013] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[Bordes *et al.*, 2014] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data: Application to word-sense disambiguation. *Machine Learning*, 94:233–259, 2014.

[Chaurasiya *et al.*, 2022] Deepak Chaurasiya, Anil Surisetty, Nitish Kumar, Alok Singh, Vikrant Dey, Aakarsh Malhotra, Gaurav Dhama, and Ankur Arora. Entity alignment for knowledge graphs: progress, challenges, and empirical studies. *arXiv preprint arXiv:2205.08777*, 2022.

[Chen *et al.*, 2020] Zhe Chen, Yuehan Wang, Bin Zhao, Jing Cheng, Xin Zhao, and Zongtao Duan. Knowledge graph completion: A review. *IEEE Access*, 8:192435–192456, 2020.

[Cheng *et al.*, 2022] Kewei Cheng, Jiahao Liu, Wei Wang, and Yizhou Sun. Rlogic: Recursive logical rule learning from knowledge graphs. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 179–189, 2022.

[Dettmers *et al.*, 2018] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI conference on artificial intelligence*, volume 32, 2018.

[Jain *et al.*, 2021] Nitisha Jain, Jan-Christoph Kalo, Wolf-Tilo Balke, and Ralf Krestel. Do embeddings actually capture knowledge graph semantics? In *European Semantic Web Conference*, pages 143–159, 2021.

[Ji *et al.*, 2015] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *International Joint Conference on Natural Language Processing*, pages 687–696, 2015.

[Karypis and Kumar, 1998] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[Kazemi and Poole, 2018] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. *Advances in neural information processing systems*, 31, 2018.

[Kochsiek and Gemulla, 2021] Adrian Kochsiek and Rainer Gemulla. Parallel training of knowledge graph embedding models: a comparison of techniques. *Proceedings of the Very Large Data Base Endowment*, 15(3):633–645, 2021.

[Lerer *et al.*, 2019] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems*, 1:120–131, 2019.

[Li *et al.*, 2022a] Ren Li, Yanan Cao, Qiannan Zhu, Guanqun Bi, Fang Fang, Yi Liu, and Qian Li. How does knowledge graph embedding extrapolate to unseen data: a semantic evidence view. In *AAAI Conference on Artificial Intelligence*, volume 36, pages 5781–5791, 2022.

[Li *et al.*, 2022b] Zhifei Li, Yue Zhao, Yan Zhang, and Zhaoli Zhang. Multi-relational graph attention networks for knowledge graph completion. *Knowledge-Based Systems*, 251:109262, 2022.

[Lin *et al.*, 2015] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI conference on artificial intelligence*, volume 29, 2015.

[Liu *et al.*, 2021] Xiyang Liu, Huobin Tan, Qinghong Chen, and Guangyan Lin. Ragat: Relation aware graph attention network for knowledge graph completion. *IEEE Access*, 9:20840–20849, 2021.

[Meilicke *et al.*, 2019] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, pages 3137–3143, 2019.

[Miller, 1995] George A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[Newman, 2006] Mark EJ Newman. Modularity and community structure in networks. *National Academy of Sciences*, 103(23):8577–8582, 2006.

[Priyadarshi and Kochut, 2021] Amitabh Priyadarshi and Krzysztof J. Kochut. Wawpart: workload-aware partitioning of knowledge graphs. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 383–395. Springer, 2021.

[Puja *et al.*, 2013] Jay Puja, Hui Miao, Lise Getoor, and William W Cohen. Ontology-aware partitioning for knowledge graph identification. In *Automated knowledge base construction*, pages 19–24, 2013.

[Schlichtkrull *et al.*, 2018] Michael Schlichtkrull, Thomas Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC*, pages 593–607. Springer, 2018.

[Sheikh *et al.*, 2022] Nasrullah Sheikh, Xiao Qin, Berthold Reinwald, and Chuan Lei. Scaling knowledge graph embedding models for link prediction. In *Proceedings of the 2nd European Workshop on Machine Learning and Systems*, pages 87–94, 2022.

[Surisetty *et al.*, 2022] Anil Surisetty, Deepak Chaurasiya, Nitish Kumar, Alok Singh, Gaurav Dhama, Aakarsh Malhotra, Ankur Arora, and Vikrant Dey. Reps: Relation, position and structure aware entity alignment. In *Companion Proceedings of the Web Conference 2022*, pages 1083–1091, 2022.

[Surisetty *et al.*, 2023] Anil Surisetty, Aakarsh Malhotra, Deepak Chaurasiya, Sudipta Modak, Siddharth Yerramsetty, Alok Singh, Liyana Sahir, and Esam Abdel-Raheem. Study of topology bias in gnn-based knowledge graphs algorithms. In *IEEE International Conference on Data Mining Workshops on Machine Learning on Graphs (ICDMW-MLoG)*, pages 1149–1156, 2023.

[Traag *et al.*, 2019] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9:5233, 2019.

[Vashishth *et al.*, 2019] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2019.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[Wang *et al.*, 2014] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI conference on artificial intelligence*, volume 28, 2014.

[Wang *et al.*, 2017] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[Wang *et al.*, 2022a] Jiang Wang, Filip Ilievski, Pedro Szekely, and Ke-Thia Yao. Augmenting knowledge graphs for better link prediction. *arXiv preprint arXiv:2203.13965*, 2022.

[Wang *et al.*, 2022b] Yun-Cheng Wang, Xiou Ge, Bin Wang, and C-C Jay Kuo. Greenkgc: A lightweight knowledge graph completion method. *arXiv preprint arXiv:2208.09137*, 2022.

[Wu *et al.*, 2020] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[Ye *et al.*, 2022] Zi Ye, Yogan Jaya Kumar, Goh Ong Sing, Fengyan Song, and Junsong Wang. A comprehensive survey of graph neural networks for knowledge graphs. *IEEE Access*, 10:75729–75741, 2022.

[Zamini *et al.*, 2022] Mohamad Zamini, Hassan Reza, and Minou Rabiei. A review of knowledge graph completion. *Information*, 13(8):396, 2022.

[Zhang *et al.*, 2020] Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *AAAI conference on artificial intelligence*, volume 34, pages 3065–3072, 2020.

[Zheng *et al.*, 2020] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 739–748, 2020.

[Zhou *et al.*, 2017] Xiaofei Zhou, Qiannan Zhu, Ping Liu, and Li Guo. Learning knowledge embeddings by combining limit-based scoring loss. In *ACM on Conference on Information and Knowledge Management*, pages 1009–1018, 2017.