LargeEA: Aligning Entities for Large-scale Knowledge Graphs

Congcong Ge, Xiaoze Liu, Lu Chen, Yunjun Gao
College of Computer Science
Zhejiang University, Hangzhou, China
{gcc,xiaoze,luchen,gaoyj}@zju.edu.cn

Baihua Zheng
School of Computing and Information Systems
Singapore Management University, Singapore
bhzheng@smu.edu.sg

ABSTRACT

Entity alignment (EA) aims to find equivalent entities in different knowledge graphs (KGs). Current EA approaches suffer from scalability issues, limiting their usage in real-world EA scenarios. To tackle this challenge, we propose LargeEA to align entities between large-scale KGs. LargeEA consists of two channels, i.e., structure channel and name channel. For the structure channel, we present METIS-CPS, a memory-saving mini-batch generation strategy, to partition large KGs into smaller mini-batches. LargeEA, designed as a general tool, can adopt any existing EA approach to learn entities' structural features within each mini-batch independently. For the name channel, we first introduce NFF, a name feature fusion method, to capture rich name features of entities without involving any complex training process; we then exploit a name-based data augmentation to generate seed alignment without any human intervention. Such design fits common real-world scenarios much better, as seed alignment is not always available. Finally, LargeEA derives the EA results by fusing the structural features and name features of entities. Since no widely-acknowledged benchmark is available for large-scale EA evaluation, we also develop a large-scale EA benchmark called DBP1M extracted from real-world KGs. Extensive experiments confirm the superiority of LargeEA against state-of-the-art competitors.

PVLDB Reference Format:
Congcong Ge, Xiaoze Liu, Lu Chen, Baihua Zheng, Yunjun Gao. LargeEA: Aligning Entities for Large-scale Knowledge Graphs. PVLDB, 15(2): 237-245, 2022.
doi:10.14778/3489496.3489504

1 INTRODUCTION

A knowledge graph (KG) consists of various entities and relations. KGs are the backbone of many real-world knowledge-driven applications, such as semantic search [47] and recommendation systems [55]. Since real-world KGs (e.g., YAGO3 [24]) are known to be highly incomplete, how to expand KGs to improve the quality of the knowledge-driven applications becomes increasingly important. EA is a prerequisite for expanding the coverage of a unified KG. It aims to find entities from two KGs that refer to the same real-world object, according to the following three steps: (i) taking two input KGs and collecting *seed alignment*; (ii) training an EA model guided by the seed alignment; and (iii) aligning the equivalent entities between the two input KGs based on the trained EA model.

Proceedings of the VLDB Endowment, Vol. 15, No. 2 ISSN 2150-8097.
doi:10.14778/3489496.3489504

Existing solutions to EA mainly rely on the structural features of entities [26, 32, 36]. They assume that the neighbors of two equivalent entities in KGs are equivalent as well [23]. Besides, recent studies have shown that incorporating *side information* of KGs (e.g., entity names [54]) facilitates the structure-based EA [23, 31], as equivalent entities usually have similar side information. Existing EA methods have demonstrated considerable performance on several representative benchmarks (such as DBP15K [33] and IDS100K [37]). However, we find out that they suffer from scalability issues. They cannot effectively align entities in real-life KGs that are much larger than the existing benchmarks. For example, among all the popular EA benchmarks, the largest KG contains only $100,000$ entities. [57] has indicated that current EA methods, when handling $100,000$ entities, either (i) require a huge memory space or (ii) have low efficiency. Nevertheless, the magnitude of real-world KGs is much larger. For instance, a real-world KG YAGO3 includes $\sim$17 million entities, while one of the largest existing benchmarks IDS100K [37] only extracts $\sim$0.6% entities from YAGO3.

To scale up the EA methods for aligning entities in large-scale KGs, a prevalent approach is to train them on a cluster of machines. Nonetheless, *the cost of training an EA model on a cluster of machines is prohibitory.* First, it is unaffordable for many users to purchase a cluster of machines. Second, it is challenging for ordinary users to deal with cluster management and unpredictable emergencies [17]. Third, distributed EA necessitates collecting the training results from different machines, and such overhead is not negligible.

The obstruction with distributed computing provokes us to partition an EA dataset into multiple mini-batches and train the samples in each mini-batch independently. It greatly saves the hardware cost as a stand-alone machine equipped with a GPU is able to run the EA model when the input dataset (i.e., a mini-batch) is of small or moderate size. Furthermore, it requires *zero* coordination among multiple machines since all the training results are stored locally. Despite these benefits, aligning entities for large-scale KGs in a mini-batch fashion, however, is still a challenging endeavor.

Challenge I: *How to effectively generate mini-batches?* A straightforward method is to partition the entire dataset into several random subsets. Because of its simplicity, it is a common practice used in various tasks, e.g., word translation [18] and text classification [14]. In those tasks, a dataset can be randomly divided into several mini-batches due to the mutual independence between data. On the contrary, EA is highly related to the structures of KGs. Random partition destroys KG's structure and thus adversely affects the EA results, as verified in the experiments to be presented in Section 3.4. Apart from the importance of maintaining the structure of each KG when generating mini-batches, it is equally crucial to allocate the possibly equivalent entities to the same mini-batch. If two equivalent entities are placed into different mini-batches, they cannot be aligned. However, maintaining the graph structure and meanwhile allocating the potentially equivalent entities to the same mini-batch is very challenging, as demonstrated by Example 1.1.

Example 1.1. Figure 1 depicts two KGs (i.e., KG_{EN} and KG_{FR}) containing equivalent entities. Each KG is divided into two mini-batches. Entities highlighted in the same color should be assigned to the same mini-batch. KG_{EN} and KG_{FR} are heterogeneous. It is worth noting that there is a wide range of highly heterogeneous KGs in real-life. Two partition strategies are used to generate mini-batches, represented by red dotted lines and blue dotted lines respectively. The *red dotted line* indicates that each KG is divided by minimizing edge-cut to minimize the structural loss. Due to the graph heterogeneity, some equivalent entities are assigned to different mini-batch in this case, such as "T-Minus (producer)" of KG_{EN} and "T-Minus" of KG_{FR}. The *blue dotted line* symbolizes that each KG is divided by preserving the equivalent entities into the same mini-batch. Nonetheless, many edges are cut, leading to the loss of structural features and poor EA results.

Challenge II: *How to recoup the loss of accuracy inevitably caused by the mini-batch generation?* Since different real-world KGs are heterogeneous, even a perfectly designed mini-batch generation method will inevitably lose seeds or destroy KG's structures, thereby reducing the EA performance, as mentioned in Example 1.1. Meanwhile, it is widely acknowledged that the *name information* of entities undoubtedly improves the EA performance [23, 38, 54]. Also, [57] has stated that several real-life entities are difficult to be aligned based purely on their structural features but are easy to be matched by their name information. In addition, it is common that real-life entities from different KGs (e.g., YAGO and DBpedia) share the same naming convention, which further makes it practical to utilize entities' name information for EA. The power of name information inspires us to explore that whether the use of the entity name could complement the seeds or the alignment features for mini-batch training. The existing name-based EA methods [10, 23, 45, 54] tend to use a pre-trained language model (e.g., BERT [10]) to initialize entity embeddings with their name features and then fine-tune these informative embeddings. As mentioned before, training an EA model with large-scale KGs is challenging, making it impractical to fine-tune the name-based entity embeddings for large-scale EA. Accordingly, we are required to exploit the name features to facilitate large-scale EA in an efficient and lightweight way.

To address these challenges, we propose LargeEA to align entities between large-scale KGs. LargeEA consists of two pivotal channels: (i) *structure channel*, which is introduced to learn the structural features of entities in a mini-batch fashion; and (ii) *name channel*, which is presented to efficiently augment the alignment results based on the entities' name features. Thereafter, LargeEA fuses both the structural features and the name features to produce the final EA results. Our contributions are summarized as follows:

- *Large-scale EA framework.* LargeEA [3] is the first EA framework that aligns entities between large-scale KGs by fusing features from *structure channel* and *name channel*. Any EA model suffering from scalability issue can be easily integrated into LargeEA to deal with large-scale EA (Section 2.1).
- *Memory saving EA channels.* In the *structure channel*, we propose a memory saving *METIS-CPS* strategy to support mini-batch training under the premise of minimizing the loss of structural features and seed alignment (Section 2.2). In the *name channel*, we introduce *NFF* and a *name-based data augmentation* to make effective use of name features without any complex training process (Section 2.3).
- *Large-scale EA benchmark.* Since no public large-scale EA benchmark is available, we develop DBP1M [3], a large-scale EA benchmark extracted from real-world KGs (Section 3).
- *Extensive experiments.* We conduct comprehensive experimental evaluation on EA tasks against state-of-the-art approaches over both the existing EA benchmarks and newly proposed DBP1M. Extensive experimental results demonstrate the superiority of LargeEA (Section 3).

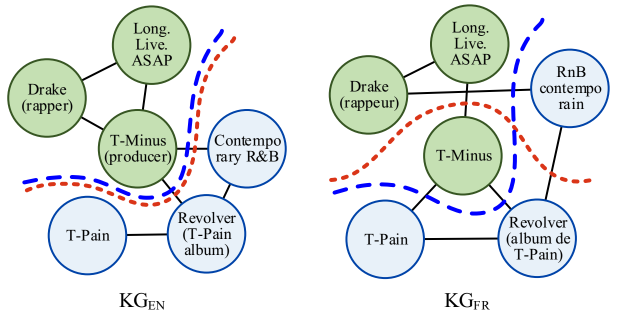


Figure 1: Example of mini-batch generation for EA

2 OUR FRAMEWORK

2.1 Problem Statement and LargeEA Overview

A knowledge graph (KG) can be denoted as $G = (E, R, T)$, where E is the set of entities, R is the set of relations, and $T = \{(h, r, t) \mid h, t \in E, r \in R\}$ is the set of triples, each of which represents an edge flowing from an entity h to another entity t via a relation r. Entity alignment (EA) [37] aims to find the 1-to-1 mapping of entities ψ from a source KG $G_s = (E_s, R_s, T_s)$ to a target KG $G_t = (E_t, R_t, T_t)$. Formally, $\psi = \{(e_s, e_t) \in E_s \times E_t \mid e_s \equiv e_t\}$, where $e_s \in E_s, e_t \in E_t$, and $\equiv$ means an equivalence relation between e_s and e_t. In most cases, a small set of equivalent entities $\psi' \subset \psi$ is known beforehand and can be used as seed alignment (training data). A representative experimental study [37] has indicated that, using a small set of seed alignment (e.g., 20% of the total number of aligned entities) as training data conforms to the real-world.

We summarize the main components of the framework LargeEA in Figure 2 to provide an overview. LargeEA takes as inputs a source KG G_s and a target KG G_t, and performs EA with the help of the *structure channel* and the *name channel*. In the structure channel, both G_s and G_t are divided into K subgraphs via the proposed mini-batch generation method METIS-CPS. It is designed to increase entity locality so that most entities can find their equivalence within the same mini-batch. After generating mini-batches, we use an EA model to learn the structural similarities between entities in each mini-batch locally. In the name channel, LargeEA presents a name feature fusion approach (NFF) to evaluate the name similarity between entities. Besides, we use a simple but highly effective *data augmentation* to generate pseudo seeds, which can complement the loss of seeds caused by the mini-batch generation process in the structure channel. Finally, we further fuse the structural similarity and the name similarity between entities to derive the EA results.

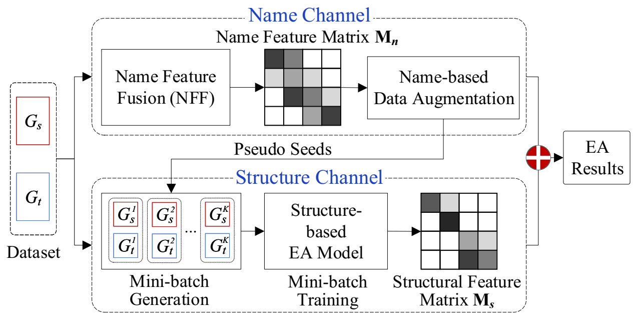

Name Channel
Name Feature Matrix $\mathbf{M}_n$
Name Feature Fusion (NFF)
Name-based Data Augmentation
G_s
G_t
Dataset
Pseudo Seeds
EA Results
Structure Channel
G_s^1 G_s^2 ... G_s^K
G_t^1 G_t^2 ... G_t^K
Structure-based EA Model
Mini-batch Generation
Mini-batch Training
Structural Feature Matrix $\mathbf{M}_s$

Figure 2: LargeEA framework

2.2 Structure Channel

2.2.1 Mini-batch Generation. Training an EA model in a mini-batch fashion is associated with the following two conditions: (i) partitioning a large-scale KG into K subgraphs; and (ii) placing every entity and its potential equivalence in the same mini-batch. To generate mini-batches under these conditions, we introduce two strategies, i.e., the vanilla partition strategy (VPS) and the METIS-based collaborative partition strategy (METIS-CPS).

VPS. It first allocates the seed alignment ψ' into K mini-batches equally and then randomly adds the remaining entities from G_s and G_t into the K mini-batches. The time and space complexities of VPS are both $O(|E_s| + |E_t|)$. VPS ensures that each mini-batch contains the same number of seeds, as an imbalanced distribution of seeds leads to poor training performance of the EA model in some mini-batches, e.g., it is unable to train the EA model without any seed alignment in a mini-batch.

METIS-CPS. Although simple, VPS relies on random partitioning, which may destroy the structure of each KG. To this end, we present METIS-CPS. The workflow is shown in Figure 3.

We first review the main idea of METIS [15], from which our strategy is derived. METIS aims to divide a graph into K subgraphs that obey the *modularity maximization principle*. Following this principle, METIS guarantees that the sum of the weights of the edge-cuts is minimized. Thus, each entity and its neighbors can be clustered into the same partition to a large extent, while irrelevant entities are located in different partitions. For simplicity, we denote the weight of an edge between entities e_i and e_j as $w(e_i, e_j)$, and suppose all the edges in a KG share an equal weight. In the current implementation, we set every $w(e_i, e_j) = 1$. The time complexity and space complexity of METIS are $O(|E| + |T| + Klog(K))$ and $O(|E| + |T|)$, respectively [48].

We then detail the partition process of METIS-CPS in the following. First, we deploy METIS [15] to split G_s into K subgraphs G_s^i, $i \in \{1, 2, ..., K\}$. We denote L_s^i the set of entities belonging to the seed alignment ψ' that are contained in G_s^i. Take Figure 3 as an example. After performing METIS, G_s is partitioned into 2 subgraphs, i.e., G_s^1 and G_s^2. We have $L_s^1 = \{e_s^1, e_s^2, e_s^3\} \in G_s^1$ and $L_s^2 = \{e_s^4, e_s^5, e_s^6\} \in G_s^2$. Next, we explain how to partition G_t into K subgraphs $G_t^{i'}$, $i' \in \{1, 2, ..., K\}$. Let $L_t^{i'}$ be the set of entities in G_t that are equivalent to the entities in L_s^i. It is preferable that entities in $L_t^{i'}$ are all included by $G_t^{i'}$. To achieve this goal, we re-assign appropriate weights to the edges in G_t in two phases.

Phase 1: Increasing weight for relevant entities. According to the modularity maximization principle, entities connected by high weighted edges in a dense graph are not likely to be partitioned. As a result, a plausible intuition for preventing entities to be divided into different mini-batches is that, we can generate a *connected graph* CG^i with high weighted edges for entities $L_t^{i'}$. Specifically, given a set of entities $L_t^{i'}$ whose equivalent entities L_s^i belong to the same subgraph G_s^i, we randomly select q entities from $L_t^{i'}$, denoted as $Q = \{e_1, e_2, ..., e_q\}$, and make sure all those q entities are able to reach all the other entities in $(L_t^{i'} - Q)$ by adding a *virtual edge* between each $e_i \in Q$ and each $e_j \in L_t^{i'} - \{e_i\}$ iff there is no edge between them to make the connected graph CG^i much denser. Thereafter, we reset the weight of each edge of CG^i to $w' \gg 1$ to prevent the entities of $L_t^{i'}$ from being assigned to different mini-batches. Since the time cost of the mini-batch generation depends on q, we set $q = 1$ in the implementation to save time. This is because empirically $q = 1$ is able to achieve satisfactory partition results in our experiments. Note that, the virtual edge is only used to assist the METIS-CPS in graph partition but not to change the graph structure of the original KG. In Figure 3, since all the equivalent entities of $e_t^{1'}$, $e_t^{2'}$, and $e_t^{3'}$ belong to $L_s^1 \in G_s$, we need to put the three entities into the same subgraph to avoid the destroy of seed alignment. Thus, we add a virtual edge between $e_t^{1'}$ and $e_t^{3'}$ to form a connected graph CG^1, and re-assign the weight of edges in CG^1 to w'. Therefore, $e_t^{1'}$, $e_t^{2'}$, and $e_t^{3'}$ are unlikely to be partitioned into different subgraphs. The time complexity of this phase is $O(|\psi'| + \frac{1}{K} \times |\psi'|^2)$.

Phase 2: Reducing weight for irrelevant entities. Let $(e_s^i, e_t^{i'})$ and $(e_s^j, e_t^{j'})$ denote two seed alignments, respectively. Here, $e_s^i, e_s^j \in E_s$ and $e_t^{i'}, e_t^{j'} \in E_t$. Assume that e_s^i and e_s^j are located in different subgraphs after partitioning, and there is an edge between $e_t^{i'}$ and $e_t^{j'}$. It is possible that the entities $e_t^{i'}$ and $e_t^{j'}$ are assigned into the same subgraph by graph partitioning. Accordingly, it is required to prevent them from being gathered into the same subgraph to guarantee that those seed alignments are well preserved even after partitioning. To achieve this purpose, a simple but effective method is to assign zero weight to the edge, i.e., $w(e_t^{i'}, e_t^{j'}) = 0$. In Figure 3, we are required to assign $w(e_t^{1'}, e_t^{4'}) = 0$, $w(e_t^{3'}, e_t^{5'}) = 0$, and $w(e_t^{3'}, e_t^{6'}) = 0$. After performing this phase, we are ready to divide the target KG G_t into subgraphs by executing the METIS strategy. The time complexity of this phase is $O\left(\frac{(K-1)|\psi'|^2}{K^2} + |E_t| + |T_t| + Klog(K)\right)$.

Finally, each mini-batch can be generated by putting together a subgraph of G_s and another subgraph of G_t that contain the most number of seed alignments. The total time complexity and space complexity of METIS-CPS are $O(|\psi'| + \frac{(2K-1)|\psi'|^2}{K^2} + |E_s| + |E_t| + |T_s| + |T_t| + Klog(K))$ and $O(|E_s| + |E_t| + |T_s| + |T_t|)$, respectively.

2.2.2 Mini-batch Training. LargeEA treats mini-batch training as a black box and users have the flexibility to utilize any existing EA model to learn a set of embeddings that can be used to represent the structural features of entities. For the EA task, many GNN-based methods [19, 22, 23, 36] have achieved promising performance by propagating the alignment signal to the entity's neighbors. Inspired by this, we propose to incorporate GNN-based models into LargeEA.

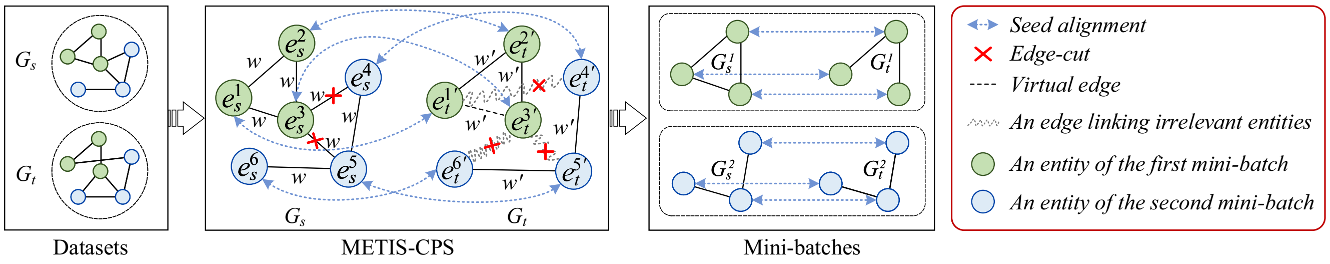


Figure 3: A toy example of METIS-CPS workflow. For simplicity, we only depict how entities contained in the seed alignment are partitioned, although all entities in the two input KGs are actually involved in this workflow.

Our current implementation includes two representative GNN-based EA models, i.e., GCN-Align [42] and RREA [26].

For ease of understanding, we sketch the core idea of how LargeEA employs the existing GNN-based EA models. Generally, the GNN-based models generate each entity's embedding as follows [16, 26, 40]. Formally, $\boldsymbol{h}^{l-1}_{\mathcal{N}^e_{e_i}} = f(\{\boldsymbol{h}^{l-1}_{e_k}, \forall e_k \in \{e_i\} \cup \mathcal{N}^e_{e_i}\})$; $\boldsymbol{h}^l_{e_i} = \sigma(\boldsymbol{W}^{l-1} \cdot \boldsymbol{h}^{l-1}_{\mathcal{N}^e_{e_i}})$. Here, $f(\cdot)$ is to aggregate information from the neighbors of every entity; $\mathcal{N}^e_{e_i}$ represents the set of neighboring entities around e_i; $\boldsymbol{W}^{l-1}$ is the transformation matrix of layer $l-1$; and $\boldsymbol{h}^l_{e_i}$ denotes the embedding of e_i after performing l-layer GNNs.

To maximize the similarities of equivalent entities in each mini-batch, GNN-based EA models often use *triplet loss*. Formally, $\mathcal{L} = \sum_{(e^i_s, e^{i'}_t) \in \psi'} \left[f_p(\boldsymbol{h}_{e^i_s}, \boldsymbol{h}_{e^{i'}_t}) + \gamma - f_n(\boldsymbol{h}_{e^i_s}, \boldsymbol{h}_{e^{i'}_t}) \right]_+$. Here, $\boldsymbol{h}_{e^i_s}$ and $\boldsymbol{h}_{e^{i'}_t}$ represent the embeddings of e^i_s and $e^{i'}_t$ learned by a structure-based EA model, respectively; $f_p(\cdot,\cdot)$ represents the distance between $\boldsymbol{h}_{e^i_s}$ and $\boldsymbol{h}_{e^{i'}_t}$; $f_n(\cdot,\cdot)$ denotes the distance of a negative entity pair derived from e^i_s and $e^{i'}_t$, generated by replacing either $\boldsymbol{h}_{e^i_s}$ or $\boldsymbol{h}_{e^{i'}_t}$ with a new embedding according to the nearest neighbor sampling [26]; $[x]_+ = max\{0, x\}$; and $\gamma > 0$ is a margin hyper-parameter.

We denote $\boldsymbol{M_s}$ the total structure-based entity similarity matrix, where each value is computed by the Manhattan distance. $\boldsymbol{M_s}$ is highly sparse. With independent mini-batch training, all non-zero similarity values lie on the diagonal blocks of $\boldsymbol{M_s}$. It saves memory cost for coping with large-scale EA. The memory cost of storing $\boldsymbol{M_s}$ is $O(|E_s|)$, i.e., the number of entities in the source KG. The time and space complexities of the entire mini-batch training process are $O(|\psi'| \times (|T_s| + |T_t|))$ and $O(|D_{str}| \times (|E_s| + |E_t|) + |T_s| + |T_t|)$, respectively. Here, $|D_{str}|$ denotes the dimension of every entity embedding learned by the mini-batch training.

2.3 Name Channel

In this section, we first present a name feature fusion method; we then introduce a name-based data augmentation; and we finally describe how to fuse name channel and structure channel for EA.

Name Feature Fusion. As performing the independent training within each mini-batch inevitably causes the loss of certain structural features of KGs, it is essential to incorporate other procedures to complement the loss caused by graph partitioning. In this work, we propose to consider entities' name features as an effective approach, namely NFF, to improve the EA performance. Given two entity sets $E_s \in G_s$ and $E_t \in G_t$, NFF computes the name similarities between E_s and E_t by fusing the name features from both *semantic aspect* $\boldsymbol{M_{se}}$ and *string aspect* $\boldsymbol{M_{st}}$. Mathematically, $\boldsymbol{M_n}$ $(= \boldsymbol{M_{se}} + \gamma \boldsymbol{M_{st}})$ represents the fused matrix, where $\gamma \in (0, 1]$ is a hyper-parameter controlling the contribution of the string-based name similarity to $\boldsymbol{M_n}$. In the current implementation, we set $\gamma = 0.05$ since many studies [20, 27] have argued that semantic feature is much more important than string-based feature. In the following, we detail SENS and STNS, the two functions to get name similarity according to the above two aspects, respectively.

Function SENS is to get semantic name similarity. Concretely, we use BERT [1] to transform each entity name into a sequence of tokens. Thereafter, the semantic embedding of each entity can be generated by applying *max-pooling*, which assigns each token an embedding with fixed-dimension for each entity and then picks the maximum value in each dimension among all embedded tokens (related to the entity) to form a new embedding representing the entity. Let $\boldsymbol{S_s}/\boldsymbol{S_t}$ denote the semantic embedding matrix of E_s/E_t. Each embedding $\boldsymbol{h_e}$ can be normalized by the equation $\boldsymbol{h_e} = \frac{\boldsymbol{h_e}}{\|\boldsymbol{h_e}\|_2 + \epsilon}$, where $\boldsymbol{h_e}$ is an entity embedding from $\boldsymbol{S_s}$ or $\boldsymbol{S_t}$, and $\epsilon > 0$ is to prevent the denominator from being zero. Since the embeddings of name features are mutually independent, we can randomly split the semantic embedding matrix ($\boldsymbol{S_s}$ or $\boldsymbol{S_t}$) into K segments for saving memory. Then, we iteratively find the top-k semantic similar entity pairs (denoted as $\boldsymbol{M^{ij}_{se}}$) between any two segments $\boldsymbol{S^i_s}$ and $\boldsymbol{S^j_t}$, where $i, j \in \{1, 2, ..., K\}$, by Faiss [13], an efficient similarity search method with GPU(s) that can cope with large-scale data. Here, we use Manhattan distance to measure the semantic similarity. Finally, the complete semantic matrix $\boldsymbol{M_{se}}$ can be obtained by only collecting the semantic similarity results computed by Faiss. The time and space complexities of SENS are $O(|D_{se}| \times |E_s| \times |E_t|)$ and $O(|D_{se}| \times (|E_s| + |E_t|))$, respectively. Here, $|D_{se}|$ denotes the dimension of every entity embedding obtained from BERT. Though the time complexity seems high, the GPU-based Faiss greatly speeds up the similarity computation process.

We would like to highlight that it is essential to consider the top-k, but not all, pairs of entities with high similarity scores for the EA task. Specifically, entity pairs with low similarity scores are probably the erroneous alignment and they are *not* expected to provide any useful information for the name-based EA task. In addition, filtering out the low scores of entity pairs but retaining the top-k similarity scores in the similarity matrix $\boldsymbol{M_{se}}$ notably reduces

Table 1: Statistics of the datasets used in experiments

Datasets		#Entities	#Relations	#Triples
IDS15K	EN-FR	15,000-15,000	267-210	47,334-40,864
	EN-DE	15,000-15,000	215-131	47,676-50,419
IDS100K	EN-FR	100,000-100,000	400-300	309,607-258,285
	EN-DE	100,000-100,000	381-196	335,359-336,240
DBP1M	EN-FR	1,877,793-1,365,118	603-380	7,031,172-2,997,457
	EN-DE	1,625,999-1,112,970	597-241	6,213,639-1,994,876

the memory cost from $O(|E_s||E_t|)$ to $O(k|E_s|)$, with $k \ll |E_t|$. Here, $O(|E_s||E_t|)$ refers to the cost of storing the similarity scores of all the candidate entity pairs (in total $|E_s||E_t|$), which is clearly impractical for large-scale KGs.

Function STNS is to measure the string-based name similarity (i.e, Levenshtein distance in the implementation) between entities. It is well-known that calculating Levenshtein distance for large-scale entity pairs is time-consuming and computationally expensive. Given two sets of entities E_s and E_t, the time and space complexities of Levenshtein-distance-based similarity computation are both $O\left(|E_s| \times |E_t| \times Max(len_{E_s}) \times Max(len_{E_t})\right)$ in general. Here, $Max(len_{E_s})$ (w.r.t. $Max(len_{E_t})$) denotes the maximum length of an entity's name from the set E_s (w.r.t. E_t). Our solution is to filter out the pairs that are extremely different, as entity pairs with different names are less likely to be aligned. Motivated by the efficiency of the datasketch library [2] for finding similar entities per entity, we deploy the datasketch library for this purpose. Concretely, the datasketch library employs MinHash-LSH to reduce the computational cost of finding similar entity pairs. The time complexity of datasketch is $O(|E_s|)$. We only retain the entity pairs P whose Jaccard similarities are above θ, the lower bound of the string difference. Then, we compute the string similarity for each entity pair in P by Levenshtein distance. We denote the string-based similarity matrix as $\boldsymbol{M_{st}}$. Similarly, the benefits of using the threshold θ are to (i) save the memory space of storing $\boldsymbol{M_{st}}$ and (ii) reduce the total number of entity pairs that require Levenshtein distance computation.

Name-based Data Augmentation. Recall that some seed alignments in the training data may be missing after mini-batch generation. To this end, we describe how we apply *data augmentation* (DA) based on the similarity between entities' name features to generate seed alignment automatically. We are inspired by the idea of *cycle consistency* from word translation [6], which states that, if two sentences from different languages can be translated to each other, they have the same meaning. Therefore, we generate pseudo seed alignment in accordance with the constraints that two entities are mutually the most similar to each other.

Channel Fusion for Aligning Entities. As highlighted by massive prior studies [23, 53, 54, 57], the name feature and structural feature can complement each other in the task of EA. Motivated by this, we fuse the name similarity matrix $\boldsymbol{M_s}$ and the structure similarity $\boldsymbol{M_n}$ derived from name channel and structure channel, respectively. To balance the importance of both structure channel and name channel, we combine $\boldsymbol{M_s}$ and $\boldsymbol{M_n}$ with equal weights, and derive the final similarity matrix between E_s and E_t, denoted as $\boldsymbol{M}$. Formally, $\boldsymbol{M} = \boldsymbol{M_s} + \boldsymbol{M_n}$. Though simple and intuitive, this approach effectively fuses the features from both name and structure aspects. We will demonstrate the effectiveness of channel fusion via experimental study to be presented in Section 3.3.

3 EXPERIMENTS

In this section, we conduct extensive experiments to verify the effectiveness and efficiency of LargeEA, using six datasets, i.e., (i) four small-scale datasets provided by the state-of-the-art benchmark IDS [37]; and (ii) two large-scale datasets newly generated by us.

3.1 Experimental Settings

Datasets and Evaluation Metrics. We conduct experiments on datasets with different scales from two cross-lingual EA benchmarks, i.e., IDS [37] and DBP1M. Table 1 lists the detailed statistics.

(i) *IDS.* Recent work [37] indicates that several EA benchmarks (e.g., DBP15K [33] and DWY100K [34]) contain much more high-degree entities than real-world KGs do. Consequently, they generate IDS, which contains four cross-lingual datasets, i.e., English and French (IDS15K$_{EN-FR}$ and IDS100K$_{EN-FR}$), and English and German (IDS15K$_{EN-DE}$ and IDS100K$_{EN-DE}$).

(ii) *DBP1M.* We create two large-scale cross-lingual datasets extracted from a well-known real-world KG, i.e., DBpedia [4]. Concretely, we retrieve ~1M ground truth of EA by utilizing the inter-language links (ILLs) and owl:sameAs among DBpedia's multilingual versions, i.e., English and French (DBP1M$_{EN-FR}$), and English and German (DBP1M$_{EN-DE}$). Unlike the IDS benchmark that ensures the number of entities from one KG is equivalent to that from another KG, we allow KGs to have different number of entities. For example, the English KGs of our proposed DBP1M benchmark contains more entities. This conforms to the real-world KGs since the English version of DBpedia is more complete than the versions in other languages. To simulate the real-world EA scenarios, we also inject *unknown entities*, which cannot find any equivalence based on the EA ground truth, into every dataset of DBP1M. This is because, it is common that only partial entities can find their equivalence in an EA dataset in real life. Specifically, we add *unknown entities* that have at least 5 entities (each of which has its equivalent entity in corresponding ground truth datasets) in their neighborhood into each KG, following Sun et al. [33].

We follow [37] to use 20% as training data, which conforms to the real world. The remaining data (80%) is to test the EA performance. We use Hits@N (N=1, 5, H@N for short), Mean Reciprocal Rank (MRR), running time (in seconds for small-scale datasets and hours for large-scale ones), and the maximum GPU Memory cost (*Mem.* for short, in GB) as the evaluation metrics. Here, the running time means the training time of every EA approach. Higher Hits@N and MRR indicate better performance. We use the NVIDIA Nsight Systems to monitor the usage of GPU memory.

Competitors. We compare LargeEA with several widely used EA models, which have presented promising EA performance: (i) *GCNAlign* [42], an attribute-powered EA model that uses vanilla GCNs and entity attributes to learn entity embeddings for alignment; (ii) *MultiKE* [56], a side-information-based EA model that unifies multiple views of entities; (iii) *RDGCN* [44], an EA model that first uses entity names to initialize entity embeddings and then learns these embeddings via a relation-aware dual graph convolutional network; (iv) *RREA* [26], a GNN-based EA model that leverages relational reflection transformation to obtain relation specific embeddings for each entity; and (v) *BERT-INT* [38], which leverages BERT [10] to discover the semantic features contained in the side information of entities instead of considering the graph structure of KGs.

Table 2: Overall EA results on IDS15K and IDS100K

Methods	IDS15K$_{EN-FR}$					IDS15K$_{EN-DE}$					IDS100K$_{EN-FR}$					IDS100K$_{EN-DE}$				
	H@1	H@5	MRR	Time	Mem.	H@1	H@5	MRR	Time	Mem.	H@1	H@5	MRR	Time	Mem.	H@1	H@5	MRR	Time	Mem.
GCNAlign	33.8	58.9	0.45	20.29	0.13G	48.1	67.9	0.57	21.60	0.13G	23.0	41.2	0.32	1225.03	1.00G	31.7	48.5	0.40	1639.49	1.00G
MultiKE	74.9	81.9	0.78	290.58	6.52G	75.6	80.9	0.78	350.85	10.52G	62.9	68.0	0.66	1277.73	16.08G	66.8	71.2	0.69	1765.08	16.08G
RDGCN	75.5	85.4	0.80	554.31	8.02G	83.0	89.5	0.86	739.72	8.02G	64.0	73.2	0.68	2852.70	16.02G	72.2	79.4	0.76	3511.14	16.02G
RREA	80.8	96.3	0.88	139.34	4.07G	85.8	96.8	0.91	137.08	4.07G	–	–	–	–	–	–	–	–	–	–
BERT-INT	94.2	96.4	0.95	969.61	14.07G	93.5	95.0	0.94	1044.28	14.07G	92.0	94.4	0.93	6991.65	14.07G	90.8	93.3	0.92	6999.01	14.07G
LargeEA-G$_{\text{EN}\rightarrow\mathbb{L}}$	88.4	92.2	0.90	77.00	1.54G	89.2	93.4	0.91	74.81	1.54G	83.9	87.5	0.86	465.10	1.74G	85.6	89.1	0.87	465.26	1.74G
LargeEA-G$_{\mathbb{L}\rightarrow\text{EN}}$	89.9	92.9	0.91	75.86	1.54G	90.8	94.2	0.92	75.89	1.54G	84.7	87.8	0.86	450.29	1.74G	85.8	89.2	0.87	452.48	1.74G
LargeEA-R$_{\text{EN}\rightarrow\mathbb{L}}$	88.7	91.9	0.90	95.43	1.54G	89.2	94.0	0.91	96.32	1.54G	84.4	88.0	0.86	552.84	4.04G	83.4	86.7	0.85	574.00	4.04G
LargeEA-R$_{\mathbb{L}\rightarrow\text{EN}}$	89.8	92.7	0.91	98.33	1.54G	91.1	94.9	0.93	96.31	1.54G	84.3	87.5	0.86	559.51	4.04G	86.4	89.6	0.88	577.88	4.04G

[1] The symbol "–" indicates that the EA model is **NOT** able to perform the EA task by using the GPU in the experimental conditions because of the memory limitation.
[2] The results of all the competitors are obtained by our re-implementation with their publicly available source code.
[3] $\mathbb{L}$ represents the non-English language. For instance, EN $\longrightarrow$ $\mathbb{L}$ denotes that the language of source KG is English and that of target KG is non-English.

Table 3: Overall EA results on DBP1M

Methods	DBP1M$_{EN-FR}$					DBP1M$_{EN-DE}$				
	H@1	H@5	MRR	Time	Mem.	H@1	H@5	MRR	Time	Mem.
LargeEA-G$_{\text{EN}\rightarrow\mathbb{L}}$	51.8	58.3	0.55	3.36	6.61G	55.3	60.8	0.58	2.59	4.59G
LargeEA-G$_{\mathbb{L}\rightarrow\text{EN}}$	50.6	56.5	0.53	3.39	8.00G	55.5	61.3	0.58	2.63	5.36G
LargeEA-R$_{\text{EN}\rightarrow\mathbb{L}}$	52.8	58.7	0.56	3.58	21.15G	56.1	61.3	0.59	2.88	16.01G
LargeEA-R$_{\mathbb{L}\rightarrow\text{EN}}$	51.5	57.0	0.54	3.71	21.17G	56.2	61.8	0.59	2.91	16.01G

Variants of LargeEA. Note that both GCNAlign and RREA provide variants that purely utilize the structural features to align entities. The former includes a vanilla GCN to learn structural features. The latter provides a GNN-based model that achieves the state-of-the-art EA performance. Since LargeEA can be easily integrated with structural-based EA models, we present two versions of LargeEA, i.e., LargeEA-G that includes the variant of GCNAlign and LagreEA-R that incorporates the variant of RREA.

Implementation Details. We detail the hyper-parameters used in LargeEA as follows. All the hyper-parameters are set without special instructions. *In the name channel*, we set the string-based similarity threshold $\theta = 0.5$ and the semantic-based similarity threshold $\phi = 50$ in NFF. Also, we fix the dimension of every entity embedding obtained by BERT (i.e., D_{se}) to be 768. *In the structure channel*, we set the number of mini-batches $K = 5$ for IDS15K, $K = 10$ for IDS100K, and $K = 20$ for our DBP1M dataset by default. Unless explicitly specified, we use RREA as the default EA model in the structure channel and optimize it with Adam for 100 epochs in each mini-batch. Besides, following the settings in [10, 42], we set the dimension of every entity embedding generated by the structure channel $|D_{str}| = 200$ for LargeEA-G and $|D_{str}| = 100$ for LargeEA-R. All experiments were conducted on a personal computer with an Intel Core i9-10900K CPU, an NVIDIA GeForce RTX3090 GPU and 128GB memory. The programs were all implemented in Python.

3.2 Overall Performance

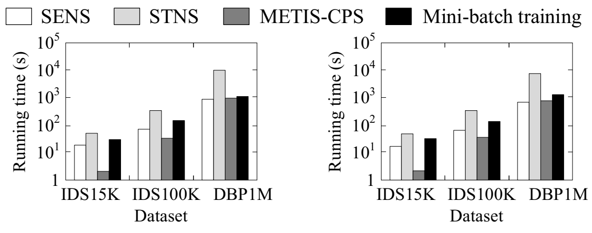
3.2.1 Performance on IDS.

Table 2 summarizes the EA performance on IDS15K and IDS100K. We first focus on the **accuracy evaluation** for the two variants of LargeEA and its competitors. First, both variants of LargeEA perform better than the existing EA models that also explore both name features and structural features. It validates the superiority of the way how the name feature and structural feature are fused in our framework. Second, compared to BERT-INT, which achieves state-of-the-art accuracy on the small-scaled datasets (i.e., IDS15K and IDS100K), LargeEA gains up to **9x GPU memory saving**. The reason is that BERT-INT highly relies on BERT [10] that has a much more complex model structure and needs to store more parameters in the training process. Moreover, BERT-INT suffers from scalability issue due to the models' inherent characteristics. Different from the other competitors that mainly utilize GPU for model training, the complex model design of BERT forces BERT-INT to store a part of parameters into RAM; otherwise, BERT-INT cannot run successfully because of limited GPU memory. To be more specific, for IDS15K, BERT-INT requires 14GB GPU memory and 7GB CPU memory on average. For IDS100K, BERT-INT requires 14GB GPU memory and 58GB CPU memory on average. Although the usage of GPU memory seems to be stable on datasets with different scales (since we set the same number of mini-batches for BERT-INT on both IDS15K and IDS100K datasets, as suggested by its original paper), the bottleneck of BERT-INT is the tremendous CPU memory requirement. As expected, it needs at least 580GB RAM for handling a large-scale dataset (e.g., DBP1M) whose size is more than 10 times that of IDS100K. To further verify the above observation, we have tried to run BERT-INT on DBP1M but failed even with a 128GB of RAM. Third, we can observe that the influence of the source KG selection on H@1 varies from 0.1% to 3%. Since different KGs are heterogeneous, it is common that selecting different KGs as sources leads to different EA accuracy (up to 4.7% on some existing models), as confirmed by DGMC [11].

We now turn our attention to the **running time evaluation**. We can observe that the training of LargeEA (both variants) is faster than that of other existing EA models. Particularly, LargeEA runs more than **10x faster** than BERT-INT, the state-of-the-art EA model. Recall that BERT-INT is extremely complex in the model structure. It is time-consuming to obtain reliable parameters in the process of training such a complex model. This shows the superiority of LargeEA in terms of running time. One exception is the performance of GCNAlign in IDS15K benchmark. The running time of LargeEA-G is larger than that of the GCNAlign alone. The reason is that LargeEA-G effectively captures the entity name's features, which requires additional running time. Despite the fast running time of GCNAlign in the small-scale IDS15K benchmark, LargeEA-G accelerates the running time up to 3.5x than GCNAlign when performing EA in IDS100K, a relatively larger benchmark.

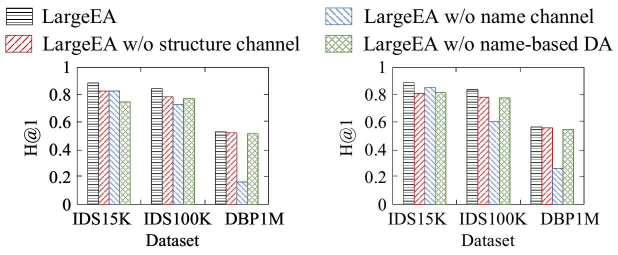
3.2.2 Performance on DBP1M.

Table 3 reports the overall EA performance of LargeEA-G and LargeEA-R on DBP1M. Note that the EA results produced by LargeEA's competitors are not reported, as



(a) EN-FR (b) EN-DE

Figure 4: Scalability analysis vs. datasize



(a) EN-FR (b) EN-DE

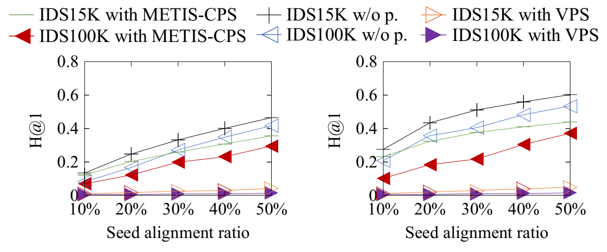
Figure 5: Ablation studies

all the competitors fail to perform EA task for any dataset of the large-scale DBP1M in the experiments. The main reason is that they all require a substantial amount of memory space. In contrast, it is observed that LargeEA is capable of scaling the current EA models to deal with large-scale entity alignment tasks. As discussed in Section 2.2.2, the reason is that the mini-batch training of LargeEA greatly saves the memory cost.

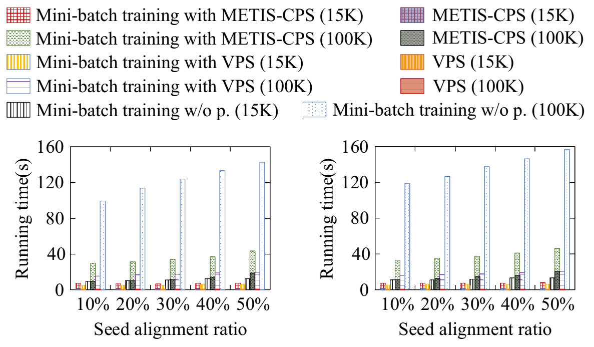
3.2.3 Scalability Evaluation. To further investigate the scalability of our LargeEA framework, we evaluate the running time of each channel of LargeEA on datasets with different scales. For the structure channel, we report the running time of computing entities' semantic name similarities (SENS) and that of computing entities' string-based name similarities (STNS), respectively. For the structure channel, we report the running time of mini-batch generation strategy (i.e., METIS-CPS) and that of EA model training. As depicted in Figure 4, we observe that the running time of each component increases almost linearly as dataset size grows. This confirms the scalability of LargeEA.

3.3 Ablation Study

We conduct ablation studies on all datasets, with results plotted in Figure 5. By removing the structure channel, the accuracy of LargeEA drops on $H@1$. This verifies that the structure channel is an indispensable part of LargeEA. We also observe that the structural channel has less influence for EA on DBP1M, compared to IDS. It is attributed to the following two reasons. First, the different number of entities from each side easily leads to more heterogeneous KGs, compared to the IDS where the source KG and the target KG share the same number of entities. Since the performance of structure-based EA methods highly relies on the graph isomorphism [11], it is more challenging to learn reliable EA signals from heterogeneous KGs, and thus results in relatively worse EA performance. Second, DBP1M contains *unknown entities*, which further exacerbate the



(a) EN-FR (b) EN-DE



(c) EN-FR (d) EN-DE

Figure 6: METIS-CPS performance vs. seed alignment

heterogeneity of the source KG and the target KG. By removing the name channel, it is observed that the accuracy decline varies from 3% to 37% on different benchmarks. This verifies that capturing name features of entities and generating pseudo seed alignment greatly improve alignment results. By removing the name-based data augmentation (DA), the accuracy decline varies from 2% to 14% on different benchmarks. In particular, the accuracy drops more significantly on IDS15K and IDS100K, compared to that on DBP1M. This is because IDS15K and IDS100K have richer structural features than DBP1M. Specifically, the name-based DA is used to provide more seeds for improving the EA results of the structural channel. The richer the structural features of a KG, the greater the improvement of EA performance that can be brought by seeds. Furthermore, EA accuracy on DBP1M is lower than that on IDS15K, attributed to the huge amount of unknown entities. Normal entities may be aligned to these unknown entities, incurring the drop of $H@1$ when evaluating the EA results according to the ground truth, which only contains known equivalent entities.

3.4 Mini-batch Generation Analysis

We explore the effect of the amount of seed alignment on METIS-CPS and VPS. We report the H@1 and the running time produced by solely using the structural channel for EA when varying the seed alignment ratio from 10% to 50%. Figure 6(a) and Figure 6(b) report the results of the alignment accuracy. First, we observe that the H@1 of both METIS-CPS and VPS improves almost linearly as the number of seed alignment increases. It is natural that more seed alignment provides more informative training signals to learn a reliable EA model. Second, it is observed that METIS-CPS performs consistently much better than VPS no matter how the number of seed alignment changes. This is because, METIS-CPS ensures that the more the seed alignment, the less destruction the KG's

Table 4: Unsupervised EA results on DBP1M

Methods	DBP1M$_{EN-FR}$					DBP1M$_{EN-DE}$				
	H@1	H@5	MRR	Time	Mem.	H@1	H@5	MRR	Time	Mem.
LargeEA-G$_{\mathbf{EN}\to\mathbb{L}}$	51.8	58.3	0.55	3.39	8.00G	55.3	60.8	0.58	2.59	4.59G
LargeEA-G$_{\mathbb{L}\to\mathbf{EN}}$	50.6	56.5	0.53	3.40	8.00G	55.6	61.3	0.58	2.64	5.37G
LargeEA-R$_{\mathbf{EN}\to\mathbb{L}}$	52.8	58.7	0.56	3.61	21.17G	56.1	61.3	0.59	2.87	21.17G
LargeEA-R$_{\mathbb{L}\to\mathbf{EN}}$	51.5	57.0	0.54	3.72	21.17G	56.2	61.8	0.59	2.93	16.01G

structure. Therefore, more structure features can be reserved to improve the EA performance produced by the structural channel. On the contrary, VPS mainly relies on random partitioning, which greatly destroys the structure of a KG and thus results in minor accuracy increase. Figure 6(c) and Figure 6(d) show the results of the running time required. As expected, the running time of VPS is shorter than that of METIS-CPS. This is because, the time complexity of VPS is lower than that of METIS-CPS, as mentioned in Section 2.2. Although VPS is faster, it greatly destroys the structure of a KG. We want to highlight that using METIS-CPS as the partition strategy can help the structural channel obtain higher alignment accuracy, which is much more critical for the task of EA.

Besides, one may be curious about the results of alignment accuracy and running time before and after the mini-batch generation. In terms of *accuracy*, as expected, the alignment accuracy after partition (w.r.t. METIS-CPS and VPS) is inferior to that before partition (w/o p. for short). This is because, graph partition inevitably cuts KG's edges and destroys the structure of a KG, thereby reducing the alignment accuracy. In terms of *running time*, it is observed that the running time of the structural channel with METIS-CPS is much shorter than that without partition. The reason is that the subgraphs within each mini-batch (after performing METIS-CPS) are much smaller than the entire KG without partition. This leads to shorter training time for learning a reliable structure-based EA model in the structural channel, compared to the training process of the structural channel without partition. To sum up, we want to emphasize the effectiveness of METIS-CPS for KG partition from the following two perspectives: (i) *Acceptable accuracy decline.* The average drop of alignment accuracy is around 8% by performing METIS-CPS, compared against that without partition. (ii) *Faster training process.* Compared with the training process without partition, performing METIS-CPS can save up to 4x training time.

3.5 Case Study: Unsupervised EA Performance

To demonstrate the superior performance of LargeEA even when seed alignment is *not* available, we present a case of applying LargeEA to conduct *unsupervised* EA on DBP1M by using the proposed data augmentation strategy to generate seed alignment automatically. Specifically, the data augmentation automatically generates 528,040 and 476,527 seeds on DBP1M$_{EN-FR}$ and DBP1M$_{EN-DE}$ respectively, with the accuracy of 93.86% and 93.85%, respectively. In this way, the proposed data augmentation can automatically generate sufficient high-quality seed alignment for EA. Table 4 reports the corresponding EA results. We can observe that LargeEA is able to achieve an accuracy that is comparable with that under supervised EA. This reflects that the name-based data augmentation can produce reliable pseudo seeds, which are able to provide positive input as real seeds. This further confirms the superiority of the proposed LargeEA for coping with real-world EA scenarios.

4 RELATED WORK

Early entity alignment (EA) methods rely on hand-crafted features [24], crowdsourcing [41, 60], and OWL semantics [12]. They are unrealistic for real-world EA scenarios with symbolic or linguistic heterogeneity. Current EA approaches find equivalent entities by measuring the similarity between the embeddings of entities.

Structures of KGs are the basis for the embedding-based EA methods. Representative EA approaches that purely rely on KGs' structures can be clustered into two categories, namely *Translational-based EA* [9, 21, 29, 30, 34, 35, 39, 58] and *GNN-based EA* [7, 19, 25, 32, 36, 42, 44, 59]. The former incorporates the translational KG embedding models (e.g., TransE [5]) to learn entity embeddings; the latter learns the entity embeddings by aggregating the neighbors' information of entities. Though GNN-based models have demonstrated their outstanding performance, they suffer from poor scalability as they highly rely on the structure of KG, as mentioned in Section 1. Therefore, LargeEA is developed to scale up these EA methods to align entities between large KGs.

Besides, lots of approaches have revealed that *side information* of KGs can facilitate the EA performance, including *entity names* [11, 23, 25, 26, 28, 33, 38, 45, 46, 49–51, 54, 56], *descriptions* [8, 38, 51, 56], *images* [22], and *attributes* [23, 33, 38, 39, 42, 43, 51, 52, 56]. They could be considered as complements to, but not competitors of, the structure-based EA models. Since every entity has its own name and the use of name information does not require any pre-processing, users/researchers tend to use the name information to promote EA. Other studies also reveal that entities' descriptions, attributes, and images contain more information than entities' name. However, these studies are either labor-intensive or error-prone and thus restrict the scope of their real-world applications. To this end, we incorporate name information in LargeEA.

5 CONCLUSIONS

In this paper, we present LargeEA to align entities between large-scale knowledge graphs. LargeEA introduces both *structure channel* and *name channel* to collaboratively align entities from large-scale KGs. In the structure channel, we propose METIS-CPS to generate multiple mini-batches and then learn the structural features of entities within each mini-batch independently. In the name channel, we explore the name features of entities from both string-based aspect and semantic aspect without any complex training process via our proposed NFF. Additionally, we exploit a name-based data augmentation to enrich the seed alignment for EA. The EA results of LargeEA are derived from the fused features of names and structures. To simulate real-world EA scenarios, we also develop a large-scale EA benchmark named DBP1M for evaluating EA performance. Considerable experimental results on EA benchmarks with different data magnitudes demonstrate the superiority of LargeEA. In the future, we would like to explore scalable EA approaches that only rely on the KG's structure, to support EA between KGs whose entities do not share the same naming convention.

ACKNOWLEDGMENTS

This work was supported by the NSFC under Grant No. 62025206, 61972338, and 62102351, and also supported by AZFT. Yunjun Gao is the corresponding author of the work.

REFERENCES

[1] The source code of BERT. https://github.com/huggingface/transformers.

[2] The source code of datasketch. https://github.com/ekzhu/datasketch.

[3] The source code of LargeEA. https://github.com/ZJU-DBL/LargeEA.

[4] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *ISWC*. 722–735.

[5] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*. 2787–2795.

[6] Richard W Brislin. 1970. Back-translation for cross-cultural research. *Journal of cross-cultural psychology* 1, 3 (1970), 185–216.

[7] Yixin Cao, Zhiyuan Liu, Chengjiang Li, Zhiyuan Liu, Juanzi Li, and Tat-Seng Chua. 2019. Multi-Channel Graph Neural Network for Entity Alignment. In *ACL*. 1452–1461.

[8] Muhao Chen, Yingtao Tian, Kai-Wei Chang, Steven Skiena, and Carlo Zaniolo. 2018. Co-training Embeddings of Knowledge Graphs and Entity Descriptions for Cross-lingual Entity Alignment. In *IJCAI*. 3998–4004.

[9] Muhao Chen, Yingtao Tian, Mohan Yang, and Carlo Zaniolo. 2017. Multilingual Knowledge Graph Embeddings for Cross-lingual Knowledge Alignment. In *IJCAI*. 1511–1517.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.

[11] Matthias Fey, Jan Eric Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. 2020. Deep Graph Matching Consensus. In *ICLR*.

[12] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. 2011. LogMap: Logic-Based and Scalable Ontology Matching. In *ISWC*. 273–288.

[13] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[14] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomás Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *EACL*. 427–431.

[15] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392.

[16] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[17] Aapo Kyrola, Guy E. Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *OSDI*. 31–46.

[18] Guillaume Lample, Alexis Conneau, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2018. Word translation without parallel data. In *ICLR*.

[19] Chengjiang Li, Yixin Cao, Lei Hou, Jiaxin Shi, Juanzi Li, and Tat-Seng Chua. 2019. Semi-supervised Entity Alignment via Joint Knowledge Embedding Model and Cross-graph Model. In *EMNLP*. 2723–2732.

[20] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60.

[21] Xixun Lin, Hong Yang, Jia Wu, Chuan Zhou, and Bin Wang. 2019. Guiding Cross-lingual Entity Alignment via Adversarial Knowledge Embedding. In *ICDM*. 429–438.

[22] Fangyu Liu, Muhao Chen, Dan Roth, and Nigel Collier. 2020. Visual Pivoting for (Unsupervised) Entity Alignment. *arXiv preprint arXiv:2009.13603* (2020).

[23] Zhiyuan Liu, Yixin Cao, Liangming Pan, Juanzi Li, and Tat-Seng Chua. 2020. Exploring and Evaluating Attributes, Values, and Structures for Entity Alignment. In *EMNLP*. 6355–6364.

[24] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. 2015. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*.

[25] Xin Mao, Wenting Wang, Huimin Xu, Man Lan, and Yuanbin Wu. 2020. MRAEA: An Efficient and Robust Entity Alignment Approach for Cross-lingual Knowledge Graph. In *WSDM*. 420–428.

[26] Xin Mao, Wenting Wang, Huimin Xu, Yuanbin Wu, and Man Lan. 2020. Relational Reflection Entity Alignment. In *CIKM*. 1095–1104.

[27] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD*. 19–34.

[28] Hao Nie, Xianpei Han, Le Sun, Chi Man Wong, Qiang Chen, Suhui Wu, and Wei Zhang. 2020. Global Structure and Local Semantics-Preserved Embeddings for Entity Alignment. In *IJCAI*. 3658–3664.

[29] Shichao Pei, Lu Yu, Robert Hoehndorf, and Xiangliang Zhang. 2019. Semi-Supervised Entity Alignment via Knowledge Graph Embedding with Awareness of Degree Difference. In *WWW*. 3130–3136.

[30] Shichao Pei, Lu Yu, and Xiangliang Zhang. 2019. Improving Cross-lingual Entity Alignment via Optimal Transport. In *IJCAI*. 3231–3237.

[31] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. 2011. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *PVLDB* 5, 3 (2011), 157–168.

[32] Zequn Sun, Muhao Chen, Wei Hu, Chengming Wang, Jian Dai, and Wei Zhang. 2020. Knowledge Association with Hyperbolic Knowledge Graph Embeddings. In *EMNLP*. 5704–5716.

[33] Zequn Sun, Wei Hu, and Chengkai Li. 2017. Cross-Lingual Entity Alignment via Joint Attribute-Preserving Embedding. In *ISWC*. 628–644.

[34] Zequn Sun, Wei Hu, Qingheng Zhang, and Yuzhong Qu. 2018. Bootstrapping Entity Alignment with Knowledge Graph Embedding. In *IJCAI*. 4396–4402.

[35] Zequn Sun, JiaCheng Huang, Wei Hu, Muhao Chen, Lingbing Guo, and Yuzhong Qu. 2019. TransEdge: Translating Relation-Contextualized Embeddings for Knowledge Graphs. In *ISWC*. 612–629.

[36] Zequn Sun, Chengming Wang, Wei Hu, Muhao Chen, Jian Dai, Wei Zhang, and Yuzhong Qu. 2020. Knowledge Graph Alignment Network with Gated Multi-Hop Neighborhood Aggregation. In *AAAI*. 222–229.

[37] 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. *PVLDB* 13, 11 (2020), 2326–2340.

[38] Xiaobin Tang, Jing Zhang, Bo Chen, Yang Yang, Hong Chen, and Cuiping Li. 2020. BERT-INT: A BERT-based Interaction Model For Knowledge Graph Alignment. In *IJCAI*. 3174–3180.

[39] Bayu Distiawan Trisedya, Jianzhong Qi, and Rui Zhang. 2019. Entity Alignment between Knowledge Graphs Using Attribute Embeddings. In *AAAI*. 297–304.

[40] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*. OpenReview.net.

[41] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[42] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. 2018. Cross-lingual Knowledge Graph Alignment via Graph Convolutional Networks. In *EMNLP*. 349–357.

[43] Zhichun Wang, Jinjian Yang, and Xiaoju Ye. 2020. Knowledge Graph Alignment with Entity-Pair Embedding. In *EMNLP*. 1672–1680.

[44] Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, Rui Yan, and Dongyan Zhao. 2019. Relation-Aware Entity Alignment for Heterogeneous Knowledge Graphs. In *IJCAI*. 5278–5284.

[45] Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, and Dongyan Zhao. 2019. Jointly Learning Entity and Relation Representations for Entity Alignment. In *EMNLP*. 240–249.

[46] Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, and Dongyan Zhao. 2020. Neighborhood Matching Network for Entity Alignment. In *ACL*. 6477–6487.

[47] Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit Semantic Ranking for Academic Search via Knowledge Graph Embedding. In *WWW*. 1271–1279.

[48] Hongteng Xu, Dixin Luo, and Lawrence Carin. 2019. Scalable Gromov-Wasserstein Learning for Graph Partitioning and Matching. In *NeurIPS*. 3046–3056.

[49] Kun Xu, Linfeng Song, Yansong Feng, Yan Song, and Dong Yu. 2020. Coordinated Reasoning for Cross-Lingual Knowledge Graph Alignment. In *AAAI*. 9354–9361.

[50] Kun Xu, Liwei Wang, Mo Yu, Yansong Feng, Yan Song, Zhiguo Wang, and Dong Yu. 2019. Cross-lingual Knowledge Graph Alignment via Graph Matching Neural Network. In *ACL*. 3156–3161.

[51] Hsiu-Wei Yang, Yanyan Zou, Peng Shi, Wei Lu, Jimmy Lin, and Xu Sun. 2019. Aligning Cross-Lingual Entities with Multi-Aspect Information. In *EMNLP*. 4430–4440.

[52] Kai Yang, Shaoqin Liu, Junfeng Zhao, Yasha Wang, and Bing Xie. 2020. COTSAE: CO-Training of Structure and Attribute Embeddings for Entity Alignment. In *AAAI*. 3025–3032.

[53] Weixin Zeng, Xiang Zhao, Jiuyang Tang, and Xuemin Lin. 2020. Collective Entity Alignment via Adaptive Features. In *ICDE*. 1870–1873.

[54] Weixin Zeng, Xiang Zhao, Wei Wang, Jiuyang Tang, and Zhen Tan. 2020. Degree-Aware Alignment for Entities in Tail. In *SIGIR*. 811–820.

[55] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *SIGKDD*. 353–362.

[56] Qingheng Zhang, Zequn Sun, Wei Hu, Muhao Chen, Lingbing Guo, and Yuzhong Qu. 2019. Multi-view Knowledge Graph Embedding for Entity Alignment. In *IJCAI*. 5429–5435.

[57] Xiang Zhao, Weixin Zeng, Jiuyang Tang, Wei Wang, and Fabian M. Suchanek. 2020. An experimental study of state-of-the-art entity alignment approaches. *TKDE* 10 (2020).

[58] Hao Zhu, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2017. Iterative Entity Alignment via Joint Knowledge Embeddings. In *IJCAI*. 4258–4264.

[59] Qiannan Zhu, Xiaofei Zhou, Jia Wu, Jianlong Tan, and Li Guo. 2019. Neighborhood-Aware Attentional Representation for Multilingual Knowledge Graphs. In *IJCAI*. 1943–1949.

[60] Yan Zhuang, Guoliang Li, Zhuojian Zhong, and Jianhua Feng. 2017. Hike: A Hybrid Human-Machine Method for Entity Alignment in Large-Scale Knowledge Bases. In *CIKM*. 1917–1926.