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.

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 model.

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. However, the magnitude of real-world KGs is much larger. For instance, a real-world KG YAGO3 includes ~17 million entities, while one of the largest existing benchmarks IDS100K [37] only extracts ~0.6% entities from YAGO3.

To scale up the EA methods for 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 challenging, as demonstrated by Example 1.1.

Example 1.1. Figure 1 depicts two KGs (i.e., KG\(_{\text{EN}}\) and KG\(_{\text{FR}}\)) 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\(_{\text{EN}}\) and KG\(_{\text{FR}}\) 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\(_{\text{EN}}\) and “T-Minus” of KG\(_{\text{FR}}\). 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, [37] 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 EA. The name-based data augmentation technique (NFF) is introduced to learn the structural similarities between entities in each mini-batch locally. In the name channel, LargeEA takes as inputs a source KG \(G_s\) and a target KG \(G_t\) to a target KG \(G_t\) \(= (E_t, R_t, T_t)\). Formally, \(\hat{\psi} = \{(e_s, e_t) \in E_s \times E_t | e_s \equiv e_t\}\), 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 \(\psi' \subset \hat{\psi}\) 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.

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 \(T = \{(h, r, t) | h, t \in E, r \in R\}\) 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 \(\psi\) from a source KG \(G_s\) to a target KG \(G_t\) \(= (E_t, R_t, T_t)\). Formally, \(\psi = \{(e_s, e_t) \in E_s \times E_t | e_s \equiv e_t\}\), 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 \(\psi' \subset \hat{\psi}\) 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 technique to generate pseudo seeds, which complements 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.
with high weight 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 $L^i_{ij}$. Specifically, given a set of entities $L^i_{ij}$ whose equivalent entities $L^i_{ij}'$ belong to the same subgraph $G^i_t$, we randomly select $q$ entities from $L^i_{ij}'$, 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 $(L^i_{ij} - Q)$ by adding a virtual edge between each $e_i \in Q$ and each $e_j \in L^i_{ij} - \{e_i\}$ 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 $L^i_{ij}$ 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 $e_1^i, e_2^i, e_3^i$, and $e_4^i$ belong to $L^i_{ij} \in G_t$, 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 $e_1^i$ and $e_2^i$ to form a connected graph $CG^i$, and re-assign the weight of edges in $CG^i$ to $w'$. Therefore, $e_1^i, e_2^i$, and $e_3^i$ are unlikely to be partitioned into different subgraphs. The time complexity of this phase is $O(|\psi'|^2 + \frac{1}{K} \times |\psi'|^2)

Phase 2: Reducing weight for irrelevant entities. Let $(e_1^i, e_2^i)$ and $(e_3^i, e_4^i)$ denote two seed alignments, respectively. Here, $e_1^i, e_2^i \in E_s$ and $e_3^i, e_4^i \in E_t$. Assume that $e_1^i$ and $e_2^i$ are located in different subgraphs after partitioning, and there is an edge between $e_3^i$ and $e_4^i$. It is possible that the entities $e_1^i$ and $e_2^i$ 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., $w(e_1^i, e_2^i) = 0$. In Figure 3, we are required to assign $w(e_1^i, e_2^i) = 0, w(e_3^i, e_4^i) = 0$, and $w(e_3^i, e_4^i) = 0$. After performing this phase, we are ready to divide the target KG $G_t$ into subgraphs by executing the METIS strategy. The time complexity is $O\left(\frac{(K-1)|\psi'|^2}{K} + |E_s| + |E_t| + Klog(K)\right)$.

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 $O(|\psi'| + \frac{(2K-1)|\psi'|^2}{K} + |E_s| + |E_t| + |T_s| + |T_t| + Klog(K))$ 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. 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, \( \mathbf{h}_{N} \) is to aggregate information from the neighbors of every entity; \( N^e_r \) represents the set of neighboring entities around \( e_i \); \( \mathbf{W}^{l-1} \) is the transformation matrix of layer \( l-1 \); and \( \mathbf{h}_{e_i} \) 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, \( \mathcal{L} = \sum_{e_i', e_i'' \in V} \left[ f_p(h_{e_i'}, h_{e_i''}) + y - f_n(h_{e_i'}, h_{e_i''}) \right]^+ \). Here, \( f_p(\cdot) \) and \( f_n(\cdot) \) represent the embeddings of \( e_i' \) and \( e_i'' \) learned by a structure-based EA model, respectively; \( f_p(\cdot) \) represents the distance between \( h_{e_i'} \) and \( h_{e_i''} \); \( f_n(\cdot, \cdot) \) denotes the distance of a negative entity pair derived from \( e_i' \) and \( e_i'' \), generated by replacing either \( h_{e_i'} \) or \( h_{e_i''} \) with a new embedding according to the nearest neighbor sampling \( [26] \); \( |x| = \max\{0, x\} \); and \( y > 0 \) is a margin hyper-parameter.

We denote \( M_s \) the total structure-based entity similarity matrix, where each value is computed by the Manhattan distance. \( M_s \) is highly sparse. With independent mini-batch training, all non-zero similarity values lie on the diagonal blocks of \( M_s \). It saves memory cost for coping with large-scale EA. The memory cost of storing \( M_s \) 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 \( O(|y'| \times (|T_s| + |T_l|)) \) and \( O(|D_{sstr}| \times (|E_s| + |E_l|) + |T_s| + |T_l|) \), respectively. Here, \( |D_{sstr}| \) 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 \( E_l \in G_l \), NFF computes the name similarities between \( E_s \) and \( E_l \) by fusing the name features from both semantic aspect \( M_{se} \) and string aspect \( M_{st} \). Mathematically, \( M_n \) \( (= M_{se} + \gamma M_{st}) \) represents the fused matrix, where \( \gamma \in (0, 1] \) is a hyper-parameter controlling the contribution of the string-based name similarity to \( M_n \). 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 \( S_e/S_t \) denote the semantic embedding matrix of \( E_s/E_l \). Each embedding \( h_e \) can be normalized by the equation \( h_e = \frac{h_e}{\|h_e\|_{\infty}} \), where \( h_e \) is an entity embedding from \( S_e \) or \( S_t \), 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 (\( S_e \) or \( S_t \)) into \( K \) segments for saving memory. Then, we iteratively find the top-k semantic similar entity pairs (denoted as \( M_{se}^{(k)} \)) between any two segments \( S'_e/S'_t \) 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 \( M_{se} \) can be obtained by only collecting the semantic similarity results computed by \( Faiss \). The time and space complexities of SENS are \( O(|D_{se}| \times |E_s| \times \left|E_l\right|) \) and \( O(|D_{se}| \times (|E_s| + |E_l|)) \), 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 \( M_{se} \) notably reduces the memory cost from \( O(|E_s||E_l|) \) to \( O(k|E_l|) \), with \( k \ll |E_l| \). Here, \( O(|E_s||E_l|) \) refers to the cost of storing the similarity.
scores of all the candidate entity pairs (in total $|E_1||E_2|$), 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 $E_1$ and $E_2$, the time and space complexities of Levenshtein-distance-based similarity computation are both $O(|E_1| \times |E_2| \times \max(len(E_1)) \times \max(len(E_2)))$ in general. Here, $\max(len(E_i))$ (w.r.t. $\max(len(E_i))$) denotes the maximum length of an entity’s name from the set $E_i$ (w.r.t. $E_i$). 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 $O(|E_i|)$. We only retain the entity pairs $P$ whose Jacard similarities are above $\theta$, 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 $M_{st}$. Similarly, the benefits of using the threshold $\theta$ are to (i) save the memory space of storing $M_{st}$ 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 $M_n$ and the structure similarity $M_s$ derived from name channel and structure channel, respectively. To balance the importance of both structure channel and name channel, we combine $M_s$ and $M_n$ with equal weights, and derive the final similarity matrix between $E_i$ and $E_j$, denoted as $M$. Formally, $M = M_s + M_n$. 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.

### 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 | 100,000-100,000 |
| EN-DE 100,000-100,000 | 381-196 | 335,359-336,240 |
| DBP1M EN-FR | 1,877,793-1,365,118 | 605-380 | 2,031,172-2,997,457 |
| EN-DE 1,625,999-1,112,970 | 597-241 | 6,213,639-1,994,876 |

### 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 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.

1. **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).

2. **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) GC-NAlign [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 uses BERT [10] to discover the semantic features contained in the side information of entities instead of considering the graph structure of KGs.
Large EA: On top of LargeEA, we also provide variants of LargeEA that incorporate the variant of GCNAlign and RREA. 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 LargeEA-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., $D_e$) 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 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 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.

3.2.2 Performance on DBP1M. Table 3 reports the overall EA performance of LargeEA-G and LargeEA-R on DBP1M. Note that the

| Methods | IDS15KEn–FR | IDS15KEn–DE | IDS100KEN–FR | IDS100KEN–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 | 65.8 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 16.08G | 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 | 33.8 58.9 0.45 20.29 0.13G | 91.1 94.9 0.93 96.31 1.54G |
| BERT-INT | 94.2 96.4 0.94 1044.28 14.07G | 93.5 95.0 0.94 137.08 14.07G | 83.8 91.0 0.87 465.10 1.74G | 85.6 89.1 0.87 465.10 1.74G |

Table 3: Overall EA results on DBP1M
We conduct ablation studies on all datasets, with results plotted in Figure 5. By removing 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 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 provides more seeds for improving the EA results of the structural channel. The richer the structural features, 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 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 seeds increases.
Table 4: Unsupervised EA results on DBP1M

| Methods | DBP1M$_{EN-FR}$ | DBP1M$_{EN-DE}$ |
|---------|-----------------|-----------------|
|         | H@1H@5 MRR Time Mem. | H@1H@5 MRR Time Mem. |
| LargeEA-C$_{EN-L}$ | 51.8 38.3 0.55 3.39 8.00G | 55.3 60.8 0.58 2.59 4.59G |
| LargeEA-C$_{EN-EN}$ | 50.6 56.5 0.53 3.40 8.00G | 55.6 61.3 0.58 2.64 5.37G |
| LargeEA-R$_{EN-L}$ | 52.8 58.7 0.56 3.61 21.17G | 56.1 61.3 0.59 2.87 21.17G |
| LargeEA-R$_{EN-EN}$ | 51.5 57.0 0.54 3.72 21.17G | 56.2 61.8 0.59 2.93 16.01G |

As real seeds. This further confirms the superiority of the proposed LargeEA for coping with real-world EA scenarios.

4 RELATED WORK

Early 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, 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 rely on any preprocessing, 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 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.
A THE INTERNAL FUNCTIONALITY OF STRUCTURE CHANNEL

To illustrate how to compute the structure-based entity similarities via the structure channel clearer, we present the pseudo-code in Algorithm 1 to outline the internal functionality of the structure channel.

Algorithm 1: Computing the structure-based entity similarities via structural channel

Input: a source KG $G_s$ and a target KG $G_t$
Output: the structural feature matrix $M_s$

1. getting pseudo seeds $\phi^s_p$ by name-based data augmentation
2. $\phi' \leftarrow \phi^s + \phi^p$
3. $(B_1, B_2, \ldots, B_K) \leftarrow$ mini-batch-generation($G_s, G_t, \phi'$)
4. foreach mini-batch $B_i$ do
   5. $H^{(i)}_s, H^{(i)}_t \leftarrow$ mini-batch-training($G_s^i \in B_i, G_t^i \in B_i$)
   6. $M_s \leftarrow$ calculating Manhattan distance between each entity pair $(h_{e_i^s}, h_{e_i^t})$
7. return $M_s$

The structural channel takes as input a source KG $G_s$ and a target KG $G_t$. First, it produces a set of pseudo seeds $\phi^s_p$ via the name-based data augmentation to enrich the seed alignment (Lines 1-2). Second, it generates $K$ mini-batches according to the proposed mini-batch generation strategies, i.e., METIS-CPS or VPS, as described in Section 2.2.1 (Line 3). Then, for each mini-batch $B_i$ (w.r.t. $G_s^i$ and $G_t^i$), it applies a GNN-based EA model to train the embeddings of entities, denoted as $H^{(i)}_s$ and $H^{(i)}_t$. The former represents the set of entity embeddings from $G_s^i$; while the latter is the set of entity embeddings from $G_t^i$ (Lines 4-5). Finally, it calculates the Manhattan distance between each entity pair $(h_{e_i^s}, h_{e_i^t})$ to form the structure-based entity similarity matrix $M_s$, where $h_{e_i^s} \in H^{(i)}_s$ and $h_{e_i^t} \in H^{(i)}_t$ (Lines 6-7).

B MINI-BATCH NUMBER EFFECT

We first compare the performance of METIS-CPS and VPS on DBP1M by varying the mini-batch number $K$ from 15 to 30. Note that $K = 15$ is the minimal mini-batch size that allows LargeEA to work on datasets in DBP1M in the experimental conditions. Since varying $K$ only affects the structure channel, we only report the accuracy of EA results (w.r.t. H@1) produced by the structure channel (i.e., LargeEA w/o name channel). We try to quantify the loss of KGs’ structures and introduce a new metric edge-cut rate ($R_{ec}$ for short) to assist the analysis of mini-batch generation performance. $R_{ec}$ is defined as the fraction of the number of edges/triples cut off to the total number of edges in the original KGs. The results are shown in Figure 7.

The first observation is that the accuracy of METIS-CPS drops as the mini-batch number $K$ increases. This is because the larger the mini-batch number, the more edge-cuts in each KGs, and the more difficult to learn reliable structural features from an EA model. The second observation is that no matter how $K$ changes, METIS-CPS always outperforms VPS on H@1. The reason is that VPS divides each KG by randomly deleting edges and hence significantly destroys the structure of a KG; while METIS-CPS is able to preserve the structural features much better. The third observation is that, the accuracy of METIS-CPS drops more slowly on DBP1M compared to the accuracy on DBP1M datasets than its accuracy on DBP1M datasets. This is because German KGs are relatively sparser than French KGs in every benchmark, and they become much sparser when $K$ ascends. Therefore, it is much harder for an EA model to learn effective structural features from a sparser KG than from a relatively denser KG. Overall, METIS-CPS consistently outperforms VPS, which demonstrates the effectiveness of METIS-CPS.

C OVERALLING MINI-BATCHES ANALYSIS

Since the mini-batches generated by METIS-CPS are disjoint with zero overlap, one may wonder whether overlapping mini-batches can affect the performance of LargeEA (with METIS-CPS for mini-batch generation). We introduce parameter $D_{ov}$ to determine how the overlapping mini-batches are generated. Specifically, given in total $K$ mini-batches, each mini-batch is combined with top-$k$ most similar mini-batches to form an overlapping mini-batch. The $K$ mini-batches are then replaced by the $K$ overlapping mini-batches generated. Note that, when $D_{ov} = 1$, each mini-batch is combined with itself, the top-1 most similar mini-batch to itself, so the generated mini-batches are still disjoint; when $D_{ov} = 3$, each mini-batch is combined with another two mini-batches that are most similar to it.

The results are shown in Figure 8. It is observed that the EA accuracy of structure channel (i.e., H@1) remains almost unchanged when $D_{ov}$ changes its values. This is because, although increasing the degree of overlap enables more equivalent entities to be
found within a mini-batch, it also introduces more invalid candidates that cannot be aligned in the mini-batch. That's why we propose the design of non-overlapping mini-batches, which introduces two benefits. First, it eliminates expensive communication with other mini-batches. Second, it reduces the number of entities and relations involved in every mini-batch, resulting in a smaller memory footprint of GPU.

### D PARTITION QUALITY ANALYSIS

As entities from different mini-batches are not able to be aligned, it's critical to ensure that our mini-batch generation algorithm is able to place equivalent entities into the same mini-batch. Accordingly, we evaluate the number of equivalent entities that are placed into the same mini-batch by adopting METIS-CPS. It is challenging to theoretically ensure that our partitioning algorithms are able to place all the equivalent entities into the same mini-batch. However, in our experimental study, the proposed METIS-CPS is able to place the considerable amount of the equivalent entities into the same mini-batch. Table 5 reports the corresponding results.

For the training set, as can be observed, the performance of METIS-CPS is inferior to that of VPS for preserving equivalent entities into the same mini-batch. This is due to the inherent characteristics of these methods. For VPS, it is designed to put the predefined equivalent entities (i.e., seed alignment) into the same mini-batch. Thus, as expected, 100% equivalent entities in the training set are correctly placed into the same mini-batch. However, METIS-CPS is designed to not only put seed alignment into the same mini-batch but also preserve the structural properties of KGs as much as possible. Recall that it is challenging to achieve the above two goals simultaneously, leading to the inferior performance of METIS-CPS in the training set.

For the test set, we observe that METIS-CPS achieves superior performance than VPS on all datasets. This verifies the effectiveness of METIS-CPS in preserving unknown equivalent entities into the same mini-batch. Specifically, recall that many studies [26, 36, 42] have illustrated that the neighbors of two known equivalent entities are likely to be equivalent. In view of this, METIS-CPS can put unknown equivalent entities into the same mini-batch to a certain extent according to both the guidance of the seed alignment and the goal of structure preservation. On the contrary, VPS purely relies on the random partition and hence there is no guarantee that unknown equivalent entities (in the test set) can be put into the same mini-batch.

We would like to highlight that putting more unknown equivalent entities in the test set is much more important than that in the training set. This is because the purpose of entity alignment is to align unknown equivalent entities correctly. If two unknown equivalent entities are placed into different mini-batches, they can not be aligned to any further extent. In addition, it is observed that the corresponding results of DBP1M are relatively inferior to the results of IDS. This is because IDS contains more structure features than DBP1M (as discussed in Section 3.3), which contribute to better KG partition results.

### E THE GPU MEMORY USAGE OF LARGEEA

We also report the GPU memory usage of the proposed LargeEA in Table 6 for further investigation, where the symbol slash (/) separates the left values from the right values. The left ones represent the results when METIS-CPS is used in the structure channel,
while the right ones correspond to results without partition. It is observed that performing structure channel usually requires more GPU memory on large dataset (i.e., DBP1M), compared to the name channel. This is because the structure channel contains a complex EA training process, which consumes a large amount of GPU memory for storing the parameters used in EA training. Differently, the name channel does not require any training process, resulting in a relatively smaller usage of GPU memory. We also observe that applying the mini-batch generation strategy significantly reduces the usage of GPU memory in the structure channel, compared to that without performing any partition. Recall that obtaining effective mini-batch generation results is challenging, as mentioned in Section 1. The results confirm the necessity of utilizing METIS-CPS for saving the GPU memory.