AALBORG UNIVERSITY

DENMARK

SEA: A Scalable Entity Alignment System

Wu, Junyang; Li, Tianyi; Chen, Lu; Gao, Yunjun; Wei, Zhiheng

Link to publication from Aalborg University

SEA: A Scalable Entity Alignment System

Junyang Wu
Zhejiang University
Hangzhou, China
wujunyang@zju.edu.cn

Tianyi Li
Aalborg University
Aalborg, Denmark
tianyi@cs.aau.dk

Lu Chen
Zhejiang University
Hangzhou, China
luchen@zju.edu.cn

Yunjun Gao
Zhejiang University
Hangzhou, China
gaoyj@zju.edu.cn

Ziheng Wei
Huawei
Hangzhou, China
ziheng.wei@huawei.com

ABSTRACT

Entity alignment (EA) aims to find equivalent entities in different knowledge graphs (KGs). State-of-the-art EA approaches generally use Graph Neural Networks (GNNs) to encode entities. However, most of them train the models and evaluate the results in a full-batch fashion, which prohibits EA from being scalable on large-scale datasets. To enhance the usability of GNN-based EA models in real-world applications, we present SEA, a scalable entity alignment system that enables to (i) train large-scale GNNs for EA, (ii) speed up the normalization and the evaluation process, and (iii) report clear results for users to estimate different models and parameter settings. SEA can be run on a computer with merely one graphic card. Moreover, SEA encompasses six state-of-the-art EA models and provides access for users to quickly establish and evaluate their own models. Thus, SEA allows users to perform EA without being involved in tedious implementations, such as negative sampling and GPU-accelerated evaluation. With SEA, users can gain a clear view of the model performance. In the demonstration, we show that SEA is user-friendly and is of high scalability even on computers with limited computational resources.

CCS CONCEPTS

• **Computing methodologies → Knowledge representation and reasoning**; *Semantic networks*.

KEYWORDS

Entity Alignment, Knowledge Graphs

1 INTRODUCTION

Knowledge graphs (KGs) represent collections of relations between real-world entities, which facilitates a wide range of downstream applications, However, KGs constructed from different sources are highly incomplete [5, 9, 13]. Entity alignment (EA) [13] has been proposed as a fundamental strategy to complete KGs. Essentially, EA aligns entities from different KGs that refer to the same real-world objects, thereby enabling the completion of KGs. Graph Neural Networks (GNNs) have attracted significant attention in recent years and GNN-based entity alignment (EA) has emerged as a promising approach to address the problem of KG incompleteness [4, 10–12, 14, 16]. Despite the advances, there remain two main challenges to be tackled.

The first challenge is how to scale up EA models to process large-scale datasets. The state-of-the-arts [10, 12] focus primarily on model design of GNNs, with little attention given to model scalability. Recent open-source EA toolkits (e.g., OpenEA [13] and EAKit [17]) have not yet provided satisfactory solutions for processing large-scale EA. However, the size of real-world KGs is much larger than that of traditional datasets used in evaluating EA models. For instance, a real-world KG YAGO3 includes 17 million entities [6].

The second challenge is how to deal with geometric problems effectively. As the size of input KGs increases, some entities tend to frequently appear as the top-1 nearest neighbors of many other entities, while outliers become isolated from the rest of the graph [5]. These issues are referred to as geometric problems [5, 13] and they can make it difficult to align entities. Although some normalization methods have been proposed to address geometric problems, they are quadratic-time method and thus may still not be feasible for very large graphs.

To address the limitations of the existing solutions and toolkits, we develop SEA – a tool aimed at facilitating the design, development, and evaluation of EA models. The key contributions of SEA are summarized as follows.

Large-scale Training. We design a novel solution to enhance the scalability of existing GNN-based. SEA first constructs mini-batches based on seed alignment, then it applies neighborhood sampling [7] on KGs to generate subgraphs, which can be fed into most GNN-based EA methods. In this way, SEA allows users to train large-scale graphs for EA on a single GPU.

Large-scale Inference and Evulation. SEA introduces an inference module to tackle the geometric problem for large-scale EA. First, the system partitions the entities set into several groups with
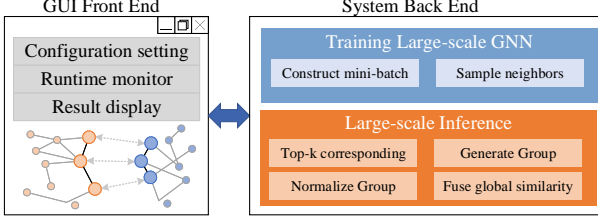
Figure 1: Architecture of SEA.

Figure 2: Training large-scale GNNs.

high entity equivalence rates and calculates the local similarity matrix for each group. Next, it normalizes the local matrices and aggregates them into a global similarity matrix. Moreover, we use FAISS [8] to create sparse matrices using top-k correspondence to speed up the evaluation process.

The Demonstration of SEA. SEA offers a suite of visualization tools to help users establish EA tasks and gain insights into the alignment process. It integrates six GNN-based models, including both the classic models [4, 14, 16] and the state-of-the-art models [10–12], making it easy for users to establish EA tasks. In addition, users can customize their GNN models flexibly using Pytorch and Deep Graph Library (DGL) [15]. The intuitive visualization interface of SEA allows users to select models and datasets, explore performance at the entity level, and compare different settings. We have created a demonstration video that can be accessed at https://github.com/Immortals88/Demo-SEA.

2 SYSTEM OVERVIEW

Figure 1 shows the architecture of SEA. SEA consists of a front end with a GUI for users and a back end that manages the EA process. The front end of the system allows users to configure EA settings, monitor the entire training process, and visualize and inspect the final EA results. The back end of the system is composed of two modules responsible for the training, inference, and evaluation of the EA process. These modules collaborate to facilitate EA across different knowledge graphs. To ease the understanding of SEA, we first introduce important preliminaries related to the entity alignment process, and then we present a detailed workflow that illustrates how users can utilize the system to perform EA.

2.1 Preliminaries

A **knowledge graph** (KG) is 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 between the head entity h to the tail entity t with the relation r.

Entity alignment (EA) aims to find the 1-to-1 mapping of entities ϕ from a source KG $G_s = (E_s, R_s, T_s)$ to a target KG $G_t = (E_t, R_t, T_t)$. $\phi = \{(e_s, e_t) \in E_s \times E_t \mid e_s \equiv e_t\}$, where $e_s \in E_s$, $e_t \in E_t$, and $\equiv$ is an equivalence relation between e_s and e_t. In most cases, a small set of equivalent entities $\phi' \subset \phi$ is known beforehand and is used as seed alignment.

GNN-based EA takes two KGs: G_s, G_t and the seed alignment ϕ' as input, and it uses graph neural networks (GNN) to (i) learn a set of embeddings for all entities E_s and E_t, denoted as $\mathbf{f} \in \mathcal{R}^{(|E_s|+|E_t|) \times D}$, where D is the dimension of embedding vectors, and (ii) maximize the similarity (e.g. cosine similarity) of entities that are equivalent in ϕ'.
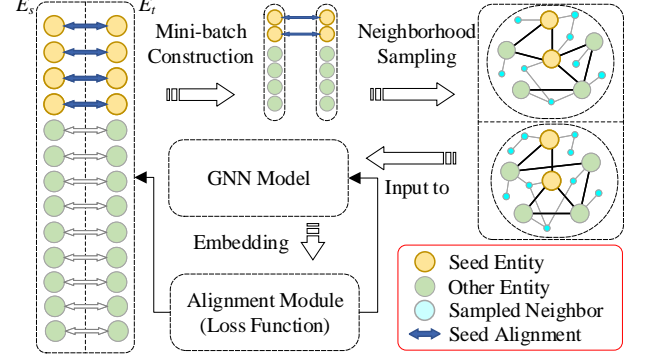
2.2 Training large-scale GNNs for EA

GNN-based EA approaches have dominated the EA tasks with promising performances by propagating the information of seed alignments to their neighbors. Formally, the embedding $h_v^{(k)}$ of an entity $v \in E$ in the k_{th} layer of GNN is obtained by aggregating localized information via $a_v^{(k)} = \text{Aggregate}^{(k)}(\{h_u^{(k-1)} \mid u \in \mathcal{N}(v)\})$ and $h_v^{(k)} = \text{Update}^{(k)}(a_v^{(k)}, h_v^{(k-1)})$. Where $h_v^{(0)} \in \mathcal{R}^D$ is a learnable embedding vector initialized randomly or generated with some language models. $\mathcal{N}(v)$ represents the set of neighboring entities around v. The model's final output on entity e is denoted as $\mathbf{f}_e$.

Most GNN-based EA approaches train their models in a whole graph manner. It is no longer feasible when the KGs become large. Since the storage of the entity embeddings and graph structure may exceed the memory capacity. To address this issue, SEA provides a general workflow for training a Siamese GNN on large graphs: (i) constructs mini-batch of entities; and (ii) samples neighborhoods in each mini-batch as a subgraph for GNN information propagation as depicted in Figures 2.

Mini-batch Construction. To scale up the existing GNN-based EA models, SEA trains these models with neighborhood sampling strategy [7]. Before we start to sample subgraph, we need to construct a bunch of entities, called mini-batch, as starting points to kick off the sampling process. Each mini-batch should keep the same setting as the whole entity set. Specifically, SEA first randomly selects a set of entities from the seed alignment that are equivalent. Next, it picks negative nodes from the entire entity set that do not overlap with seed entities. Finally, a mini-batch consisting of these two parts (positive and negative entities) is generated as shown in Figures 2.

Neighborhood Sampling.

In SEA, the entire graph information is stored in CPU memory. When a mini-batch is generated, SEA samples the k-hop neighborhoods for each entity in the mini-batch to form a subgraph. The hop number k is determined by the number of layers defined in GNN models. By controlling the size of the mini-batch, the hop number, and the fan-out (i.e. maximum sampled neighbors) for each node, the resulting subgraph is guaranteed to be small enough to fit in GPU memory. The subgraph and the embeddings of the involved entities are then shifted to the GPU for GNN training. This significantly reduces the GPU memory cost, making it possible to train large-scale inputs (e.g., DBP1M [6]).

Model Training and Loss Functions. The GNN models used in SEA take the sampled subgraphs as input and generates the embedding of the entities in the mini-batch as output. The loss
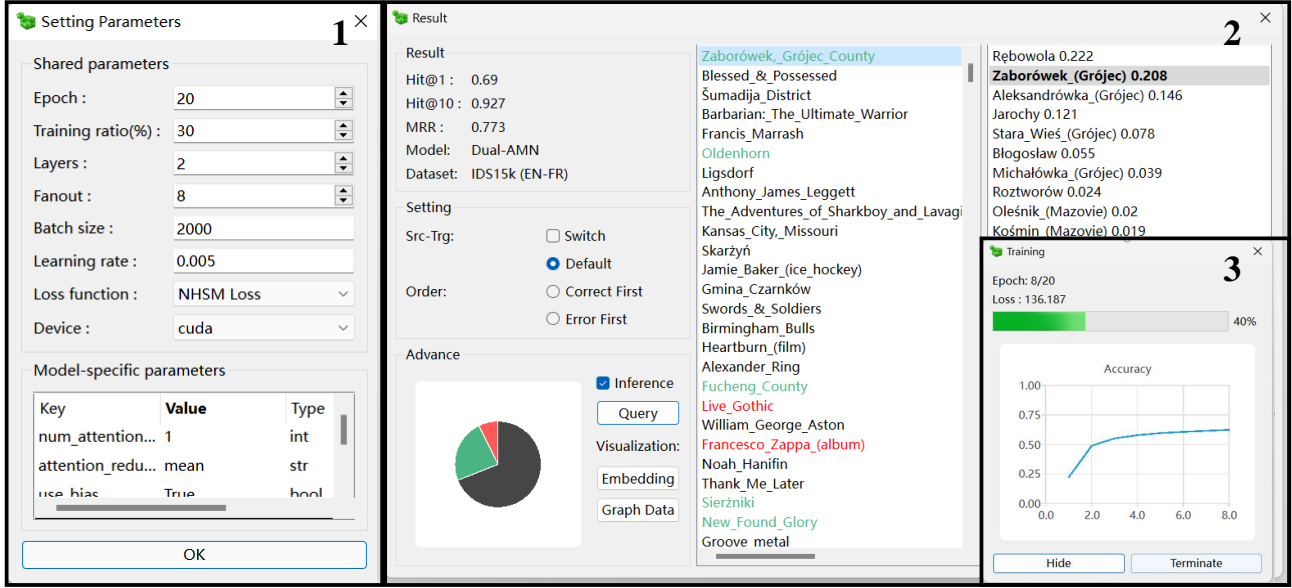
Figure 3: The Graphic User Interface of SEA.

function is designed to encourage the embeddings of equivalent entities to be close together while keeping non-equivalent entities far apart. This allows us to measure the distance between entities and identify potential alignments. In SEA, we implement multiple GNN models (cf. Section 2.4) and two commonly used loss functions: the triplet loss [16] and the normalized hard sample mining loss [10]. Note that the users can define their own loss functions.

2.3 Large-scale Inference

After obtaining the KG embeddings, we can get the similarity matrix by measuring the distances between them. However, as the size of KGs grows, the geometric problems become more severe [5]. Many existing studies [5, 9, 13] use normalization methods on the similarity matrix to alleviate the geometric problems. But they have at least quadratic time complexity. To solve the problem, SEA first groups entities that have a high likelihood of being equivalent together, and computes the local similarity matrix within each group. Next, SEA uses FAISS with GPU acceleration to calculate the top-k correspondences, which are then used to form a global sparse similarity matrix. Finally, SEA normalizes the local similarity matrices and merges them with the global sparse similarity matrix to find the alignment, greatly reducing the time complexity.

Maintaining performance while reducing complexity poses two challenges: (i) generating groups with a high entity equivalence rate to ensure a 1-to-1 mapping, and (ii) merging the local similarity matrices to ensure overall accuracy. To address these challenges, in SEA's inference stage, we adopt our previous work, ClusterEA [5]. ClusterEA trains a classifier to group entities with a high equivalent rate and proposes a SparseFusion method to normalize the global similarity matrix effectively and efficiently.

SEA offers two types of evaluation metrics to assess the performance of EA: (i) the Hits-k metric (H@k), indicating whether the ground truth appears in top-k ranking; and (ii) the Mean reciprocal rank (MRR), indicating the average of the reciprocal rank of the ground truth.

2.4 Models and Dataset

Models. In SEA, we implement the stochastic training of six GNN-based EA models using DGL [15]. These models are (i) *GCNAlign* [16], the first GNN-based EA model; (ii) *GAT* [14], a common GNN model that introduces attention technique and has been used as a building block in many recent approaches [9, 10]; (iii) *MRAEA* [11], a GNN-based EA model that models entity embeddings by integrating the node's incoming and outgoing neighbors and the connected relations' meta semantics; (iv) *MuGNN* [4], a GNN-based EA model that utilizes multi-channel information to align entities across different knowledge graphs; (v) *RREA* [12], a GNN-based EA model that leverages relational reflection transformation to obtain relation-specific embeddings for each entity; and (vi) *Dual-AMN* [10], an EA model that encompasses two attention layers to model both intra-graph and cross-graph relations.

Datasets. SEA has built-in support for five commonly used real-world datasets: DBP15K [3], DWY100K [3], SRPRS [3], IDS [3], and a large-scale dataset DBP1M [2].

3 DEMONSTRATION

SEA is a cross-platform application that utilizes PyTorch for its back-end logic and Qt for its graphic user interface. Its basic interface, shown in Figure 3, includes functionalities such as hyperparameter configuration, EA model training, alignment results evaluation, and visualization. This demonstration serves to showcase the SEA.

Hyper-parameter Configuration. SEA offers support for six real-world datasets commonly used in EA (cf. Section 2.4). Users can either choose one of the supported datasets or load a local dataset. Then, SEA checks the input format for compatibility upon loading. Once the dataset is loaded, users are required to select an EA model for training and to set the corresponding hyper-parameters through interface No.1, as shown in Figure 3. To simplify the process, some suggested hyperparameters are pre-filled based on the selected model and dataset, eliminating the need for non-experts to manually tune them.

```
1  class SimpleGCN(nn.Module):
2      def __init__(self, in_feats, out_feats):
3          super(SimpleGCN, self).__init__()
4          self.conv = dgl.nn.GraphConv(in_feats, out_feats,
5              norm='none', weight=True, bias=True)
6          xavier_normal_(self.conv.weight)
7          zeros_(self.conv.bias)
8      def forward(self, blocks):
9          in_feats = blocks[0].srcdata['feature']
10         edge_weight = blocks[0].edata['weight']
11         h = self.conv(blocks[0], in_feats,
12             edge_weight=edge_weight)
13         return h
```

Figure 4: An example code of implementing a one-layer GCN for the EA task in SEA.

The hyper-parameters in SEA are divided into two parts: (i) *Shared hyperparameters*, which must be specified by all EA models, such as learning rate, loss functions, model layer number, and training epochs; and (ii) *Model-specific parameters*, which must be set specifically for the selected model, such as the number of attention heads in Graph Attention Network (GAT) related approaches.

User-defined GNN models. SEA provides the ability to implement new GNN models for EA. Users can define their own GNN models, as depicted in Figure 4. To define a new model, a user only needs to follow the guideline of PyTorch and DGL [15] to inherit the *torch.nn.Module* class and implement two methods: *__init__()* that constructs the model and *forward(blocks)* that takes a list of sampled subgraph as input and outputs the node embedding of its destination nodes. The input of the *forward* function is a series of *DGLBlocks* [1] representing the sampled subgraph, where each one yields the node feature and edge weights of its layer. The model design is similar to building a model for node classification tasks, which simplifies the task for users by eliminating the need to implement tedious procedures.

Training of EA model. Once the parameters has been set, the user can initiate the EA task by clicking the "Run!" button. The training process, as outlined in Section 2.2, is fully parallelized on the GPU. SEA offers real-time tracking of the training loss and prediction accuracy(cf. No.2 in Figure 3). Users can further enhance tracking by integrating other visualization tools such as TensorBoard.

Results Evaluation & Visualization. When the EA process is completed, the system proceeds to the alignment results visualization stage, which contains two parts: *align result visualization* and *graph data visualization*, as shown in Figures 3 and 5, respectively.

For the *align result visualization*, we provide basic evaluated information including H@1, H@10 and MRR generated by our large-scale evaluation unit. To facilitate the interpretability of EA models, SEA depicts the EA results in the form of lists. Entities in set E_s are shown in a list on the left-hand side. If we click one entity in the list, SEA will show the top-10 corresponding entities in E_t produced by the EA model, with the ground truth highlighted (if any). If the entity is properly aligned, it is shown in black. If the top-10 correspondences include the ground truth (but not as the top choice), the entity is shown in green. If the ground truth is not found in the top-10 correspondences, the entity is shown in red. Several options are provided to re-arrange the list as depicted in Figure 3, No.2. In particular, users can switch the source and target and prioritize the erroneous entities. Moreover, SEA maintains a
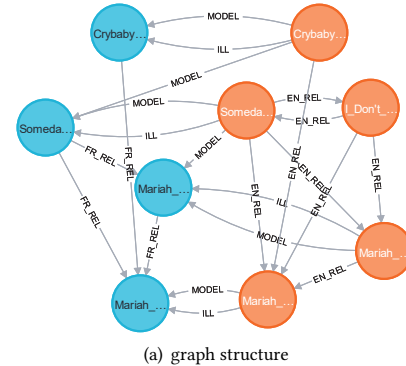
(a) graph structure
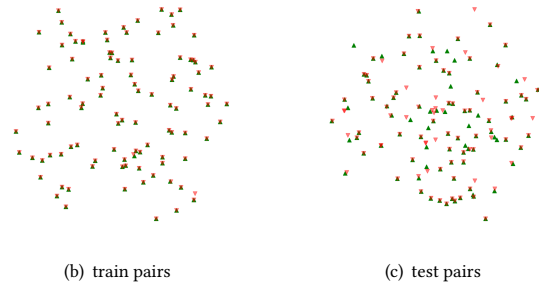
(b) train pairs   (c) test pairs

Figure 5: Graph data visualization.

class for each entity, which includes the entity's name, its neighbors, the ground truth, and a list of candidate entities generated by the model. This means users can write their own queries to select specific entities they wish to examine. As described in Section 2.3, we have proposed an inference method to address geometric problems. By clicking the "inference" checkbox, users can observe the effectiveness of this module. If the box is not checked, SEA will identify the nearest neighbor for each entity in a naive way. This leads to a drop of 20% in Hits@1 for almost all settings.

For the *graph data visualization*, SEA provides visualization tools for both KG structure and entity embeddings. To depict the two KGs and their alignment, we have integrated SEA with Neo4j, as shown in Figure 5(a). The nodes of the two graphs are painted with different colors (orange for EN and blue for FR in this example). The top-10 alignment produced by the EA model is added as edges (MODEL) between graphs, along with the ground truth edges (ILL). Users can utilize Cypher Query Language to explore the aligned result in more detail. As for embedding visualization, we use a dimension reduction method TSNE to project embedding vectors to 2D points, where the distance between points reflects their distance in the high-dimensional space. Figure 5(b) shows that the embeddings of the training data are highly aligned, while the test pairs have some minor errors. By exploring the embedding visualization, users can gain more insight into the EA process.

ACKNOWLEDGMENTS

REFERENCES

[1] dgl.to_block(). https://docs.dgl.ai/generated/dgl.to_block.html.

[2] LargeEA dataset DBP1M. https://github.com/ZJU-DBL/LargeEA.

[3] Nanjing University Websoft Group Github Page. https://github.com/nju-websoft.

[4] 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.

[5] Yunjun Gao, Xiaoze Liu, Junyang Wu, Tianyi Li, Pengfei Wang, and Lu Chen. 2022. ClusterEA: Scalable Entity Alignment with Stochastic Training and Normalized Mini-Batch Similarities. In *SIGKDD*. 421–431.

[6] Congcong Ge, Xiaoze Liu, Lu Chen, Baihua Zheng, and Yunjun Gao. 2022. LargeEA: Aligning Entities for Large-scale Knowledge Graphs. *PVLDB* 15, 2 (2022), 237–245.

[7] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1025–1035.

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

[9] Xiaoze Liu, Junyang Wu, Tianyi Li, Lu Chen, and Yunjun Gao. 2023. Unsupervised Entity Alignment for Temporal Knowledge Graphs. In *WWW*.

[10] Xin Mao, Wenting Wang, Yuanbin Wu, and Man Lan. 2021. Boosting the Speed of Entity Alignment 10×: Dual Attention Matching Network with Normalized Hard Sample Mining. In *WWW*. 821–832.

[11] 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.

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

[13] 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.

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

[15] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. In *arXiv preprint arXiv:1909.01315*.

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

[17] Kaisheng Zeng, Chengjiang Li, Lei Hou, Juanzi Li, and Ling Feng. 2021. A comprehensive survey of entity alignment for knowledge graphs. *AI Open* 2 (2021), 1–13.