The VLDB Journal (2022) 31:1143–1168
https://doi.org/10.1007/s00778-022-00747-z

SPECIAL ISSUE PAPER

A benchmark and comprehensive survey on knowledge graph entity alignment via representation learning

Rui Zhang[1] · Bayu Distiawan Trisedya[2] · Miao Li[2] · Yong Jiang[1] · Jianzhong Qi[2]

Received: 15 March 2021 / Revised: 17 January 2022 / Accepted: 26 March 2022 / Published online: 24 May 2022

Abstract
In the last few years, the interest in knowledge bases has grown exponentially in both the research community and the industry due to their essential role in AI applications. Entity alignment is an important task for enriching knowledge bases. This paper provides a comprehensive tutorial-type survey on representative entity alignment techniques that use the new approach of representation learning. We present a framework for capturing the key characteristics of these techniques, propose a benchmark addressing the limitation of existing benchmark datasets, and conduct extensive experiments using our benchmark. The framework gives a clear picture of how various techniques work. The experiments yield important results about the empirical performance of the techniques and how various factors affect the performance. One important observation not stressed by previous work is that techniques making good use of attribute triples and relation predicates as features stand out as winners. We are also the first to investigate the question of how to perform entity alignments on large-scale knowledge graphs such as the full Wikidata and Freebase (in Experiment 5).

Keywords Knowledge graph · Entity alignment · Knowledge graph alignment · Knowledge base · Representation learning · Deep learning · Embedding · Graph neural networks · Graph convolutional networks

1 Introduction

Knowledge bases are a technology used to store complex structured and unstructured information, typically facts or knowledge. A *knowledge graph* (KG), which is a knowledge base modeled by a graph structure or topology, is the most popular form of knowledge bases and has almost become a synonym of knowledge base today. There have been continuous research and development on KGs for several decades due to their significance in systems that involve reasoning based on knowledge and facts. Example KGs include open-source ones such as DBpedia [1], Freebase [3], YAGO [19], as well as proprietary ones such as those developed by Google [11] and Microsoft [13]. In the last few years, there has been an explosive growth of interest in KGs in both the research community and the industry due to their essential role in AI applications such as natural language processing (including dialogue systems/chatbots, question answering sentence generation, etc.) [61,71,74,75], search engines [23], recommendation systems [79], and information extraction [12,52].

One of the most important tasks for KGs is *entity alignment* (EA), which aims to identify entities from different KGs that represent the same real-world entity. EA enables enrichment of a KG from another complementary one, hence improving the quality and coverage of the KG, which is critical for downstream applications. Different KGs may be created via different sources and methods, so even entities representing the same real-world entity may be denoted differently in different KGs, and it is challenging to identify all such *aligned entities* accurately. Figure 1 shows a toy example of EA on two KGs $\mathcal{G}_1$ and $\mathcal{G}_2$ (each in a dashed-line rectan-

✉ Rui Zhang
rayteam@yeah.net
https://ruizhang.info/

Bayu Distiawan Trisedya
bayu.trisedya@unimelb.edu.au

Miao Li
miao4@student.unimelb.edu.au

Yong Jiang
jiangy@mail.sz.tsinghua.edu.cn

Jianzhong Qi
jianzhong.qi@unimelb.edu.au

[1] Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, China

[2] The University of Melbourne, Parkville, Australia

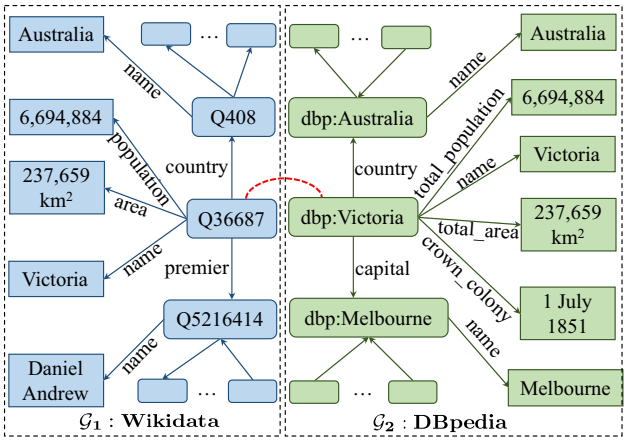

Fig. 1 An example of EA

gle), which are tiny subsets from two real publicly available KGs, Wikidata and DBpedia, respectively. The rounded rectangles represent entities and the rectangles represent attribute values. An arrow between rounded rectangles indicates a relation predicate, which forms a *relation triple*, e.g., (`dbp:Victoria, country, dbp:Australia`). An arrow between a rounded rectangle and a rectangle indicates an attribute predicate, which forms an *attribute triple*, e.g., (`dbp:Victoria, total_area, "237,659 km`^{2}"). We can see that the same real-world entity may have different surface forms in the two KGs such as `Q36687` v.s. `dbp:Victoria`. The two KGs have complementary information about this entity: $\mathcal{G}_1$ has information about its premier and $\mathcal{G}_2$ has information about its capital. The information about this entity can be enriched if we can determine that `Q36687` in $\mathcal{G}_1$ refers to the same real-world entity as `dbp:Victoria` in $\mathcal{G}_2$, i.e., `Q36687` and `dbp:Victoria` are *aligned entities*. EA between $\mathcal{G}_1$ and $\mathcal{G}_2$ is to find all the pairs of aligned entities from the two KGs. In this example, there are two pairs of aligned entities $\langle$`Q36687, dbp:Victoria`$\rangle$ and $\langle$`Q408, dbp:Australia`$\rangle$.

Traditional EA techniques use data mining or database approaches, typically heuristics, to identify similar entities. The accuracy of such approaches is limited, and heuristics is difficult to generalize. In the past several years, a very large number of studies on EA take the new approach of deep learning to learn effective vector representation (i.e., embeddings) of the KG and then performing EA based on the learned representation, which achieve much better accuracy. They also have better generalizability as they rarely rely on ad hoc heuristics. In the rest of this paper, by saying *embedding-based EA techniques* or simply *EA techniques*, we refer to those taking this new representation learning approach rather than traditional approaches unless explicitly specified otherwise. There are a few recent experimental studies aiming at benchmarking EA techniques [49,81,82]. They have high-level discussions on frameworks for embedding-based techniques and summarize a good range of EA papers, but their focus is on experimental comparison, but without self-contained explanation on each technique. Moreover, the frameworks discussed in those papers miss important mechanisms such as the use of semantic information of KGs (e.g., strings of relation predicates, attribute predicates, and attribute values), making those frameworks inapplicable to many EA techniques, especially the latest ones. In comparison with the aforementioned studies, this paper fills the void and make the following contributions:

- We provide a comprehensive *tutorial-type* survey to help readers understand how each technique works with little need to refer to individual full papers.
- We provide a comprehensive framework that accommodates almost all the embedding-based EA techniques, capturing their key components, strategies and characteristics. We also comparatively analyze different techniques in reference to the framework.
- We identify significant limitations of existing benchmark datasets such as bijection, lack of name variety, and small scale (detailed in Sect. 7.1). To address these limitations, we devise a benchmark[1] that complement the existing collection of benchmark datasets. Further, we conduct an extensive experimental study comparing the performance of the state-of-the-art techniques on our datasets.

The rest of the paper is organized as follows. Section 2 provides preliminaries, including problem formulation and a summary on traditional EA techniques. We present our framework for EA techniques in Sect. 3. Section 4 covers KG structure embedding models, mainly *translation-based* and *graph-neural-network-based* embedding, which are the foundation of embedding-based EA techniques. Sections 5 and 6 survey the most representative EA techniques based on the two major KG structure embeddings, respectively. Section 7 discusses the limitations of existing datasets, presents our proposed new datasets, and reports an extensive experimental study using our datasets. Section 8 concludes the paper and discusses future directions.

2 Preliminaries

Notation and Terminology Different notation and terminology conventions have been used in different papers in the literature. In this paper, we make a great effort at a standard notation and terminology convention that provides clarity

[1] Our benchmark and all the code for our experiments are available at https://github.com/ruizhang-ai/EA_for_KG.

Table 1 Frequently used symbols

Symbols	Descriptions
$(\mathcal{E}, \mathcal{R}, \mathcal{A}, \mathcal{V}, \mathcal{T})$	A knowledge graph
$\mathcal{E}$	A set of entities
$\mathcal{R}$	A set of relation predicates
$\mathcal{A}$	A set of attribute predicates
$\mathcal{V}$	A set of attribute values which may be numeric or literal
$\mathcal{T}$	A set of triples, which may consist of relation triples $\mathcal{T}_r$ and attribute triples $\mathcal{T}_a$
(h, r, t)	A relation triple, which consists of a head entity h, a tail entity t, the relation predicate r between the entities
$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}$	Embeddings of the head entity, relation predicate and tail entity, respectively
(e, a, v)	An attribute triple, which consists of an entity e, an attribute predicate a, and the attribute value v
$\boldsymbol{e}, \boldsymbol{a}, \boldsymbol{v}$	Embeddings of an entity, attribute predicate, and attribute value, respectively
f_{triple}	Triple score function
f_{align}	Alignment score function
σ	A nonlinear activation function
$\mathcal{S}$	A set of pre-aligned entities from $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathcal{S}'$	A set of corrupted entity alignments from $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathcal{T}_r'$	A set of corrupted relation triples
$\|$	Concatenation of two vectors

and is consistent with as many existing papers as possible. The terminology will be seen throughout the paper as various terms are introduced, and the frequently used symbols in our notation convention are summarized in Table 1.

We use bold lowercase (e.g., $\boldsymbol{e}$), bold uppercase (e.g., $\boldsymbol{M}$), and math calligraphy (e.g., $\mathcal{E}$) to denote vectors, matrices, and sets, respectively.

In the literature, a method proposed by a paper may have been referred to by different terms such as model, approach, technique, algorithm, method, etc.; we primarily use the term *technique* in this paper, while other terms might be used when the semantics are clear.

2.1 Problem formulation

We first introduce some notation. A KG denoted as $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{A}, \mathcal{V}, \mathcal{T})$, consisting of a set of entities $\mathcal{E}$, a set of relation predicates $\mathcal{R}$, a set of attribute predicates $\mathcal{A}$, and a set of attribute values $\mathcal{V}$, represents knowledge in the form of a set of triples $\mathcal{T}$. There are two types of triples, *relation triples* (denoted by $\mathcal{T}_r$) in the form of (h, r, t) and *attribute triples* (denoted by $\mathcal{T}_a$) in the form of (e, a, v); $\mathcal{T} = \mathcal{T}_r \cup \mathcal{T}_a$. A relation triple (h, r, t) indicates a relation predicate r between two entities, a head entity h and a tail entity t, where $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$. Take a triple in Fig. 1 as an example: (`dbp:Victoria, country, dbp:Australia`). Here, `dbp:Victoria.` and `dbp:Australia` are the head entity and tail entity, respectively, and `county` is the relation predicate. An attribute triple (e, a, v) indicates that an entity $e \in \mathcal{E}$ has the attribute value of $v \in \mathcal{V}$ for the attribute (predicate) $a \in \mathcal{A}$. For example, in (`dbp:Victoria, total_area, "237,659` km^2"), `total_area` is the attribute predicate and "`237,659` km^2" is the attribute value.

The problem of EA is formulated as follows.

Definition 1 (*Entity Alignment (EA)*) Given two KGs $\mathcal{G}_1 = (\mathcal{E}_1, \mathcal{R}_1, \mathcal{A}_1, \mathcal{V}_1, \mathcal{T}_1)$ and $\mathcal{G}_2 = (\mathcal{E}_2, \mathcal{R}_2, \mathcal{A}_2, \mathcal{V}_2, \mathcal{T}_2)$, EA aims to identify every pair of entities (e_1, e_2), $e_1 \in \mathcal{E}_1$, $e_2 \in \mathcal{E}_2$, where e_1 and e_2 represent the same real-world entity (i.e., e_1 and e_2 are aligned entities). □

2.2 Related problems and traditional techniques

Related Problems There have been research on various problems similar to EA on KGs.

Both sources structured. *Entity matching* [39,55], *object identification* [50], and *record linkage* [14] aim to align entities from two different relational databases, where both data sources are well structured. Solutions for these problems mostly find database records that are similar in terms of contents.

One source semi-structured and the other unstructured. *Entity resolution* [2] and *entity linking* [26] aim to match entity mentions from natural sentences, which are unstructured, to the corresponding entities in a KG, which is semi-structured. A KG is semi-structured because it consists of a graph (structured), and attributes and predicates, which are in the form of natural language or other un-predefined types (unstructured). Solutions for these problems mostly find database records that have similar contents to named entities recognized from natural sentences.

In comparison, EA on KGs aligns entities from two different KGs, both of which are semi-structured.

Traditional Techniques for EA Among traditional techniques for EA on KGs, some have focused on improving the *effectiveness* of the matching of entities via different entity similarity measures. For example, RDF-AI [15] uses fuzzy string matching based on sequence alignment, word relation, and taxonomic similarity. SILK [22] provides the Link Specification Language, which allows users to specify the similarity measures for comparing certain attributes. LD-Mapper [41] combines string similarity and entity nearest neighbors. PARIS [44] includes schema matching (e.g., classes and sub-classes of entities) to compute the entity similarity. Some other traditional EA techniques focus on the

efficiency of entity matching, e.g., LIMES [34] uses clustering to reduce the amount of similarity computation.

Traditional EA techniques, as exemplified above, usually use data mining or database approaches, typically heuristics, to identify similar entities. It is difficult for them to achieve high accuracy and to generalize.

3 Generic framework of embedding-based EA

We provide a generic framework for embedding-based EA techniques to capture key components and strategies in Fig. 2. The components drawn in dashed lines are optional. The approach of embedding-based EA typically consists of three components, an *embedding module*, an *alignment module*, and an *inference module*.

The embedding module and the alignment module may be trained alternatively or jointly, and these two together compose the *training modules* for EA.

Embedding module The embedding module aims to learn (typically low-dimensional) vector representations, i.e., *embeddings* of entities. There are four types of raw information that may be taken as input features to the embedding module: *KG structure* (in the form of relation triples in the raw KG data), *relation predicates*, *attribute predicates* and *attribute values* (attribute predicates and attribute values are grouped into one component "Attributes" in Fig. 2 and the 5th column in Table 2). The embedding module may produce as output the embeddings of entities, entity pairs, relation predicates, attributes, etc.; we refer to the process of "encoding" the input features into the targeted embeddings as *KG embedding*. Among all the possible input features, the KG structure is the most critical one. The machine learning model for embedding KG structure, which we simply term as the *KG structure embedding model*, serves as the skeleton of an EA technique, and other types of information may be optionally added to the KG structure embedding model to create a more sophisticated KG embedding. Note that the other types of information (i.e., relation predicates, attribute predicates, and attribute values) are usually in the form of strings and hence contain rich semantic information, which may greatly benefit EA as we will see.

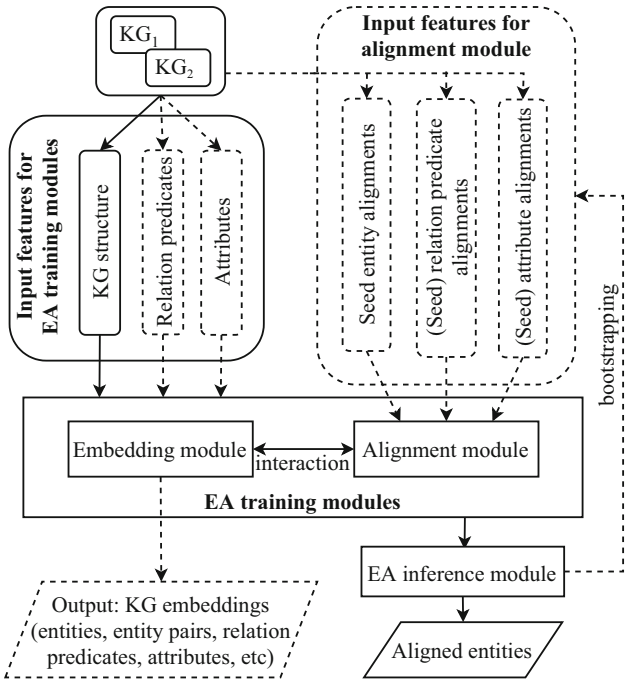

Fig. 2 Framework of embedding-based EA techniques. Dashed lines indicate optional parts

The KG structure embedding model mostly follows one of two paradigms, translation-based and GNN-based. Translation-based models mainly utilize relation triples while GNN-based models mainly utilize the neighborhood of entities. How relation triples are utilized in the translation-based models and how the neighborhood of entities is utilized in the GNN-based models are detailed in Sects. 4.1 and 4.2, respectively.

Relation predicates and attribute predicates may be encoded as categorical values or strings. Attribute values are usually encoded as strings.

Alignment module The embedding module computes the embeddings of each KG separately, which makes the embeddings of $\mathcal{G}_1$ and $\mathcal{G}_2$ fall into different vector spaces. The alignment module aims to unify the embeddings of the two KGs into the same vector space so that aligned entities can be identified, which is a major challenge for EA. EA techniques usually make use of a set of manually aligned entities, relation predicates, or attributes, called *seed alignments*, as input features to train the alignment module. The most common approach is using a set of seed *entity* alignments $\mathcal{S} = \{(e_1, e_2) | e_1 \in \mathcal{E}_1, e_2 \in \mathcal{E}_2, e_1 \equiv e_2\}$. These seeds consist of pairs of entities (e_1, e_2), where e_1 is an entity in $\mathcal{E}_1$ and e_2 is an entity in $\mathcal{E}_2$. The seeds are used to compute a loss function for the embedding module to learn a unified vector space. A typical example of how the loss function may be defined is as follows:

$$\mathcal{L} = \sum_{(e_1,e_2)\in\mathcal{S}} \sum_{(e_1',e_2')\in\mathcal{S}'} \times \max\left(0, \left[\gamma + f_{\text{align}}(\boldsymbol{e}_1, \boldsymbol{e}_2) - f_{\text{align}}(\boldsymbol{e}_1', \boldsymbol{e}_2')\right]\right) \quad (1)$$

where $\gamma > 0$ is a margin hyperparameter. The above loss function is designed to minimize the distances between pairs of entities in the seed entity alignments $\mathcal{S}$ and maximize the distances between the pairs of entities (e_1', e_2') in corrupted seed alignments $\mathcal{S}'$, which are negative samples obtained by

replacing an entity in the seed alignments with a random entity. Here, the distance between a pair of entities is computed by f_{align}, which we call the *alignment score function*. It indicates how (dis)similar two entities are; the more the two entities are aligned (i.e., similar to each other), the smaller f_{align} is. The most commonly used alignment score functions are cosine similarity, L_1 norm (i.e., Manhattan distance), and L_2 norm (i.e., Euclidean distance). We do not observe large difference in performance in our experiments when swapping these metrics (also note that L_2 norm and cosine similarity are equivalent). Some techniques customize the alignment score function to serve more sophisticated optimization goals such as Wang et al. [59]. The function $\max(0,)$ ensures that any negative margin loss values are not added to the total loss. Some techniques may exploit other types of seed alignments, including *seed relation predicate alignments*, *seed attribute predicate alignments* and *seed attribute value alignments* (seed attribute predicate alignments and seed attribute value alignments are grouped into one component "Seed attribute alignment" in Fig. 2). Relation predicate and attribute predicate alignment are needed because the same predicate may be stored in different surface forms, e.g., one KG has the attribute predicate `birth_date` while the other KG has the attribute predicate `date_of_birth`. The need for attribute value alignment is similar.

Note that like the embedding module, the alignment module may also use the four types of raw information (KG structure, relation predicates, attribute predicates, and attribute values) besides seed alignments as features. Some EA techniques may use an unsupervised method to train the alignment modul, e.g., AttrE [51] exploits attribute triples to learn a unified attribute vector space, so manually labeled seed relation predicate/attribute alignments are not necessity, and we put parenthesis on the word "seed" for relation predicate alignments and attribute alignments in Fig. 2. At least one input feature is required to train the alignment module, though.

In summary, the input features to the alignment module may be raw information such as KG structure, relation predicates, and attributes, as well as entity/relation/attribute alignments which may be created manually or automatically.

Bootstrapping is a common strategy when limited seed alignments are available. The idea is that those aligned entity/attribute/relation produced by the EA inference module are fed back to the alignment module as training data, and this process may be iterated multiple times. Note that creating seeds takes human effort, which is expensive. Bootstrapping may help reduce human effort but is at the cost of much more computation since it iterates training multiple times. From the summary in Table 2, we can see that bootstrapping is very popular among the translation-based techniques but not among the GNN-based techniques. Eight out of 15 translation-based techniques exploit bootstrapping, but only one out of 17 GNN-based technique exploits bootstrapping. The reason is that the GNN-based technique is better at capturing the relationships between entities in a graph compare to the translation-based techniques. Thus, the translation-based techniques use bootstrapping to improve their capability in capturing the entity relationships. However, we believe that bootstrapping is also helpful for GNN-based techniques, which may be investigated in future work.

EA Inference module This module aims to predict whether a pair of entities from $\mathcal{G}_1$ and $\mathcal{G}_2$ are aligned. In practice, almost all the techniques use the following alternative aim: given a target entity e_1 from $\mathcal{G}_1$, the EA inference module aims to predict an entity e_2 from $\mathcal{G}_2$ that is aligned to e_1; we may call e_1 (e_2) the aligned entity or the counterpart entity of e_2 (e_1). The aligned entity may not exist if a similarity threshold is applied.

The most common approach for the inference module is *nearest neighbor search* (NNS), which finds the entity from $\mathcal{G}_2$ that is the most similar to e_1 based on their embeddings obtained from the EA training module. Commonly used similarity measures include cosine similarity, Euclidean distance, and Manhattan distance. When describing individual techniques later, we may omit the inference module if it uses this most common approach of NNS.

The NNS inference approach may incur *many-to-one alignment*, where two different entities from a KG are aligned with the same entity from the other KG. To avoid it, some studies impose a one-to-one constraint.

Discussions Table 2 summarizes representative EA techniques according to six key characteristics (the first row): KG structure embedding, KG structure, the way attributes used as input features, the way relation predicates used as input features, input features for the alignment module, and whether bootstrapping is used. Column 2 captures the paradigm of the embedding module. Columns 3 to 5 describe all the raw input features. Column 6 describes the input features of the alignment module. Early techniques make use of fewer types of information. For example, MTransE [7] only uses KG structure as features for the embedding module and seed relation triple alignments for the alignment module. Newer techniques such as AttrE [51] and MultiKE [80] use all the four types of raw information as input features for training modules as well as various types of features for the alignment module. As our experimental study shows (cf. Table 5), techniques using more types of input features tend to have better performance.

Some studies especially early ones regard whether a technique can perform EA on multilingual KGs or only on monolingual KGs as an important distinction. We argue that this is not an essential characteristic of EA techniques.

Table 2 A summary of embedding-based EA techniques

Technique	KG structure embedding	KG structure	Relation predicates as input features	Attributes as input features	Input features for alignment module	B[a]
MTransE (2017)	TransE	Triple	–	–	Seed entity alignments; Seed relation predicate alignments	–
IPTransE (2017)	PTransE	Path	–	–	Seed entity alignments; Seed relation predicate alignments	✓
JAPE (2017)	Modified TransE[b]	Triple	–	Data type of attribute value	Seed entity alignments; Seed relation triple alignments	–
BootEA (2018)	Modified TransE[b]	Triple	–	–	Seed entity alignments	✓
KDCoE (2018)	TransE	Triple	–	Entity description	Seed entity alignments	✓
OTEA (2019)	TransE	Triple	–	–	Seed entity alignments	–
SEA (2019)	TransE	Triple	–	–	Seed entity alignments	✓
NAEA (2019)	Modified TransE[b]	Neighborhood	–	–	Seed entity alignments	✓
TransEdge (2019)	TransEdge	Triple	–	–	Seed entity alignments	✓
SX19 (2019)	TransE	Triple	–	–	Seed entity alignments; Seed relation predicate alignments	–
AKE (2019)	Modified TransE[b]	Triple	–	–	Seed entity alignments	–
AttrE (2019)	TransE	Triple	String of relation predicate	Attribute triple as relation triple; Character sequence of attribute value; String of attribute predicate	Attribute triple as relation triple; Character sequence of attribute value; String of attribute/relation predicate	–
MultiKE (2019)	Modified TransE[b]	Triple	String of relation predicate	Attribute triple as relation triple; String of attribute predicate/value; Entity name	Seed entity alignments; Relation/attribute predicate alignments	–
COTSAE (2020)	TransE	Triple	–	Character sequence of attribute value/predicate	Seed entity alignments	✓
JarKA (2020)	Modified TransE[b]	Triple	–	Word embeddings of attribute value	Seed entity alignments; Seed relation/attribute predicate alignments; Seed attribute value alignments	✓
GCN-Align (2018)	GCN	Neighborhood	–	Attribute triple as relation triple	Seed entity alignments	–
HMAN (2019)	GCN	Neighborhood	Bag-of-words of relation predicate	Bag-of-words of attribute value; Entity description	Seed entity alignments	–

Table 2 continued

Technique	KG structure embedding	KG structure	Relation predicates as input features	Attributes as input features	Input features for alignment module	B[a]
AVR-GCN (2019)	VR-GCN	Neighborhood	–	–	Seed entity alignments; Seed relation predicate alignments	–
HGCN (2019)	GCN	Neighborhood	–	Entity name	Seed entity alignments, Relation predicate alignments	–
RDGCN (2019)	DPGGNN	Neighborhood	–	Entity name	Seed entity alignments	–
GMNN (2019)	GCN	Neighborhood	–	Entity name	Seed entity alignments	–
MuGNN (2019)	GCN	Neighborhood	–	–	Seed entity alignments; Seed relation predicate alignments	–
NMN (2020)	GCN	Neighborhood	–	Entity name	Seed entity alignments	–
CG-MuAlign (2020)	CollectiveAGG	Neighborhood	–	–	Seed entity alignments	–
CEA (2020)	GCN	Neighborhood	–	Entity name	Seed entity alignments	–
SSP (2020)	GCN	Neighborhood	–	–	Seed entity alignments	–
XS20 (2020)	GCN	Neighborhood	–	Entity name	Seed entity alignments	–
KECG (2019)	GAT+TransE	Neighborhood	–	–	Seed entity alignments	–
AliNet (2020)	GAT	Path	–	–		–
AttrGNN (2020)	GAT	Neighborhood	–	String of attribute value; Entity name	Seed entity alignments	–
MRAEA (2020)	GAT	Neighborhood	String of relation predicate; Direction of relation predicate	Entity name	Seed entity alignments	✓
PEA (2020)	GAT	Neighborhood	–	String of attribute value; Entity name	Seed entity alignments	–

"–" means *not applicable*

[a]The column "B." indicates whether the technique uses bootstrapping.

[b]"Modified TransE" proposed in different papers may differ from each other

The reason is that most recently proposed techniques make use of the semantic information of KGs such as the strings of relation predicates, attribute predicates, and attribute values (cf. Table 2), and we can perform automatic translation on the semantic information into the target language so that both KGs are in the same language, and then conduct EA on monolingual KGs such as in JarKA [6]. Our experimental study (Table 7) validates this.

4 KG structure embedding models

We review two paradigms of KG structure embedding, which underlie embedding-based EA techniques.

4.1 Translation-based embedding models

The essence of translation-based embedding models is treating a relation in KGs as a "translation" in a vector space between the head and the tail entities.

TransE [4] is the first translation-based embedding model, which embeds both entities and relations into a unified (typically low-dimensional) vector space. The main idea is that, if we can find a perfect suite of vector representations, i.e., *embeddings*, of entities and relation predicates, then for any relation triple (h, r, t), the corresponding embeddings $\boldsymbol{h}$, $\boldsymbol{r}$ and $\boldsymbol{t}$ should satisfy the vector translation operation of $\boldsymbol{h} + \boldsymbol{r} = \boldsymbol{t}$. For example, the embedding of `Victoria` plus the embedding of `capital` should equal the embedding of `Melbourne`. In other words, if we define the following function

$$f_{\text{triple}}(h, r, t) = ||\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}|| \tag{2}$$

then ideally this function should have the value of 0 for the embeddings of all the true relation triples. Equation 2 is called the *triple score function*, which measures the plausibility of a relation triple (the smaller the function value, the more likely (h, r, t) form a true relation triple). In reality, the embeddings of all the entities and relation predicates are unknown and need to be learned. Further there may well not exist a suite of embeddings such that $f_{\text{triple}} = 0$ for all the relations. Therefore, the aim of learning becomes finding a suite of embeddings that minimize the sum of f_{triple} for all the relations in a KG. To learn effectively, Bordes et al. [4] use the strategy of negative sampling, i.e., for any (h', r', t') that do not form a true relation triple, $f_{\text{triple}}(h', r', t')$ should be a large value, e.g., (`Victoria`, `capital`, `dog`); these negative samples are called *corrupted relation triples* and are generated from true triples with either the head or the tail entity being replaced by a random entity. The learning process usually randomly initialize all the embeddings and then minimize a margin-loss-based objective function below via gradient descent:

$$\mathcal{L} = \sum_{(h,r,t)\in\mathcal{T}_r} \sum_{(h',r,t')\in\mathcal{T}'_r} \times \max\left(0, \left[\gamma + f_{\text{triple}}(h, r, t) - f_{\text{triple}}(h', r', t')\right]\right) \tag{3}$$

where $\gamma > 0$ is a margin hyperparameter, $\mathcal{T}_r$ is the set of true relation triples, and $\mathcal{T}'_r$ is a set of corrupted relation triples.

After the seminal work of TransE, several variants of translation-based KG structure embedding models are proposed with improvements on the embedding space [20,58,66,67] or on the triple score function [68].

Interested readers are referred to Wang et al. [57] and Ji et al. [21] for surveys on KG embedding models.

4.2 GNN-based embedding models

Graph neural networks (GNNs) have yielded strong performance on graph data analysis and gained immense popularity [65]. There are two representative models, namely graph convolutional networks (GCNs) [24], and graph attention networks (GAT) [54], which will be detailed later. These two models are frequently used in recent KG embedding and EA studies because KGs are of graph structure by nature. Unlike translation-based embedding models, which treats each triple separately, GNN-based embedding models focus on aggregating information from the neighborhood of entities together with the graph structure to compute entity embeddings. The essence of GNN-based embedding models is aggregating information from the neighborhood to a target node according to rules of *message passing* [17], i.e., the embedding information is propagated from neighbor entities to the target entity through the edges. The optimization goal of GNN-based embedding is to map entities with a similar neighborhood into embeddings close to each other in the embedding space.

Graph Convolutional Networks (GCNs) [24] compute a target node's embedding as a low-dimensional vector (i.e., embedding) by aggregating the features of its neighbors in addition to itself, following the rules of message passing in graphs. Specifically, a GCN is a multi-layer GNN denoted by a function $f(\boldsymbol{X}, \boldsymbol{A})$, where the inputs are feature vectors of a graph's nodes represented by a matrix $\boldsymbol{X}$ and the graph's adjacency matrix $\boldsymbol{A}$. The element $a_{ij} \in A$ indicates the connectivity between nodes i and j and can be viewed as the weight of the edge between the two nodes. The features of the neighbors encoded as embeddings are passed on to the target node weighted by the edge weights. This message passing process is formulated as:

$$\boldsymbol{Q}^{(l+1)} = \sigma\left(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{Q}^{(l)}\boldsymbol{W}^{(l)}\right) \tag{4}$$

where $\boldsymbol{W}^{(l)}$ is a learnable weight matrix in the lth layer, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacency matrix with self-connections, $\tilde{\boldsymbol{D}}$ is a diagonal matrix of node degrees ($\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$), $\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ is the normalization of $\tilde{\boldsymbol{A}}$ by node degrees, and $\boldsymbol{Q}^{(l+1)}$ is the output of the lth layer, which consists of the node embeddings computed by the GCN after l iterations. The input of the lth layer is $\boldsymbol{Q}^{(l)}$, which in turn is the output of the previous layer. The node embeddings are usually initialized by the input matrix, i.e., $\boldsymbol{Q}^{(0)} = \boldsymbol{X}$, and the final layer produces the final node embeddings learned by the GCN. Usually $\boldsymbol{A}$ is determined by the connectivity of the graph where 1 means connected and 0 means not; $\boldsymbol{A}$ may also be determined by heuristics such as the similarity between nodes and the values may be between 0 and 1. Once $\boldsymbol{A}$ is determined, it remains unchanged during the training which means that it is not learned.

Graph Attention Networks (GAT) [54] aggregate information from neighborhood with the *attention mechanism* [53] and allows for focusing on the most relevant neighbors. Conceptually, GAT is similar to GCN in the sense that they both perform message passing to compute the node embeddings; the main difference is that the edge weights of GCNs (i.e., the adjacency matrix) are not learned but those of GAT (i.e., the attentions) are. Specifically, GAT is also a multi-layer GNN denoted by a function $f(X)$ and the input X is the feature vectors of nodes. The output of the lth layer is computed based on the attention mechanism as follows:

$$\boldsymbol{q}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\boldsymbol{W}\boldsymbol{q}_{(j)}^{l}\right) \tag{5}$$

where $\boldsymbol{W}$ is a learnable weight matrix; $\boldsymbol{Q}^{(l+1)} = \boldsymbol{q}_1^{(l+1)}, \boldsymbol{q}_2^{(l+1)}, \ldots, \boldsymbol{q}_n^{(l+1)}$ is the output of the lth layer; n is the number of the nodes; $\boldsymbol{q}_i^{(l+1)}$ is the node embedding of node i computed by the GAT after l iterations. The input of the lth layer is $\boldsymbol{Q}^{(l)}$, which in turn is the output of the previous layer using $\boldsymbol{Q}^{(l-1)}$ as input. The node embeddings are usually initialized by the input matrix, i.e., $\boldsymbol{Q}^{(0)} = \boldsymbol{X}$, and the final layer produces the final node embeddings learned by the GAT. The attention weight α_{ij} is computed by a softmax normalization over attention coefficients:

$$\alpha_{ij} = \frac{\exp(c_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(c_{ik})} \tag{6}$$

where $\mathcal{N}_i$ indicates the set of nodes in the neighborhood of node i. The attention coefficient c_{ij} is the correlation between nodes, which is learned as follows:

$$c_{ij} = \textsc{LeakyReLU}\left(\boldsymbol{w}^T[\boldsymbol{W}\boldsymbol{q}_i^{(l)} \| \boldsymbol{W}\boldsymbol{q}_j^{(l)}]\right) \tag{7}$$

where the parameter vector $\boldsymbol{w}$ is used to transform the concatenation of two node embeddingss into a scalar.

GAT applies multi-head attention as follows:

$$\boldsymbol{q}_i^{(l+1)} = \Big\|_{k=1}^{K}\sigma\Big(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\boldsymbol{W}^k\boldsymbol{q}_j^{(l)}\Big) \tag{8}$$

where the output is the concatenation of K independent self-attentions with different normalized attention weight α_{ij}^k and weight matrix $\boldsymbol{W}^k$.

Variants The two representative GNN models, GCN and GAT, have served as the foundation for more sophisticated models designed for various applications. Alternative symmetric matrices have been proposed to replace the adjacency matrix of GCNs (e.g., AGCN [77] and DGCN [87]), and various ways of computing attention have been proposed for GAT. A comprehensive discussion on GNNs is given by Wu et al. [65].

5 Translation-based EA techniques

This section reviews representative translation-based EA techniques. We focus on the two key components, the embedding module determined by f_{triple}, and the alignment module determined by f_{align}. The KG embedding in translation-based EA techniques either use TransE [4] directly or its variants, which encodes KG structure by relation triples, paths or neighborhood. We review the techniques that only use KG structure for their KG embedding in Sect. 5.1 and the techniques that exploit other types of information, i.e., relation predicates and attributes for their KG embedding in Sect. 5.2.

5.1 Techniques that only use KG structure

MTransE [7] is the first translation-based model for embedding-based EA. Its embedding module uses TransE to embed the entities and relation predicates from each KG into a different embedding space with part of the loss function being the same as Eq. 3. To make these embeddings all fall into a unified space, the alignment module learns cross-KG transitions by minimizing the sum of the alignment score function for all the seed relation triple alignments as follows:

$$\mathcal{L} = \sum_{(tr_1,tr_2)\in\mathcal{S}_t} f_{\text{align}}(tr_1, tr_2) \tag{9}$$

where $\mathcal{S}_t$ is a set of seed relation triple alignments from the $\mathcal{G}_1$ and $\mathcal{G}_2$ (essentially the combination of seed entity alignments and seed relation predicate alignments), and $f_{\text{align}}(tr_1, tr_2)$

is the alignment score function. Different from the alignment score function described in Sect. 3, which computes the (dis)similarity of two *entities*, here the alignment score function computes the (dis)similarity of two *relation triples*, $tr_1 = (h_1, r_1, t_1) \in \mathcal{G}_1$ and $tr_2 = (h_2, r_2, t_2) \in \mathcal{G}_2$. To compute the alignment score, MTransE has three strategies to construct cross-KG transitions including *distance-based axis calibration*, *transformation vectors*, and *linear transformations*. According to their experimental study, MTransE with the linear transformation strategy has the best performance. This strategy learns a linear transformation between the entity embeddings of $\mathcal{G}_1$ and $\mathcal{G}_2$ with the following alignment score function:

$$f_{\text{align}}(tr_1, tr_2) = ||\boldsymbol{M}^e_{ij}\boldsymbol{h}_1 - \boldsymbol{h}_2|| + ||\boldsymbol{M}^r_{ij}\boldsymbol{r}_1 - \boldsymbol{r}_2|| + ||\boldsymbol{M}^e_{ij}\boldsymbol{t}_1 - \boldsymbol{t}_2|| \quad (10)$$

where $\boldsymbol{M}^e_{ij}$ and $\boldsymbol{M}^r_{ij}$ are linear transformations on entity embeddings and relation predicate embeddings, respectively. Minimizing f_{align} will minimize the distance between the transformed entities/relation predicates from $\mathcal{G}_1$ and those from $\mathcal{G}_2$, making the embeddings of the two KGs fall into the same vector space.

The inference module of MTransE uses NNS.

IPTransE [84] first learns the embeddings of $\mathcal{G}_1$ and $\mathcal{G}_2$ separately in the embedding module with an extension of TransE named *PTransE* [29]). Different from TransE, PTransE can model indirectly connected entities by considering the path between them, which is composed of relation predicates that form a translation between them. The alignment module of IPTransE learns transitions between $\mathcal{G}_1$ and $\mathcal{G}_2$ with three different strategies based on seed entity alignments: *translation-based*, *linear transformation*, and *parameter sharing*.

The translation-based strategy adapts the idea of "translation" to the cross-KG context and treats alignment as a special relation predicate $\boldsymbol{r}^{(\mathcal{E}_1 \to \mathcal{E}_2)}$ between two sets of entities, $\mathcal{E}_1$ and $\mathcal{E}_2$ from $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. The alignment score function is defined as:

$$f_{\text{align}}(e_1, e_2) = ||\boldsymbol{e}_1 + \boldsymbol{r}^{(\mathcal{E}_1 \to \mathcal{E}_2)} - \boldsymbol{e}_2|| \quad (11)$$

where $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are the embeddings of two entities $e_1 \in \mathcal{E}_1$ and $e_2 \in \mathcal{E}_2$. The objective function is a weighted sum of the loss function of PTranE and f_{align} on seed entity alignments.

The linear transformation strategy learns a transformation matrix $\boldsymbol{M}^{(\mathcal{E}_1 \to \mathcal{E}_2)}$, which makes two aligned entities close to each other, with the alignment score function below:

$$f_{\text{align}}(e_1, e_2) = ||\boldsymbol{M}^{(\mathcal{E}_1 \to \mathcal{E}_2)}\boldsymbol{e}_1 - \boldsymbol{e}_2|| \quad (12)$$

The objective function is a weighted sum of the loss function of PTranE and f_{align} on seed entity alignments.

The parameter sharing strategy forces $\boldsymbol{e}_1 = \boldsymbol{e}_2$, which indicates that a pair of aligned entities share the same embedding, and hence applying f_{align} on two aligned seed entities always gives 0. The objective function reduces to the loss function of PTranE. The parameter sharing strategy shows the best joint embedding learning performance among the three strategies.

In the training process, IPTransE adopts bootstrapping and has two strategies to add newly aligned entities to the seeds: a hard strategy and a soft strategy. Other techniques usually apply the hard strategy where newly-aligned entities are directly appended into the set of seed alignments, which may suffer from error propagation. In the soft strategy, reliability scores are assigned to newly aligned entities to mitigate error propagation, which correspond to the embedding distance between aligned entities. This may be implemented as a loss item added to the objective function.

BootEA [46] models EA as a one-to-one classification problem and the counterpart of an entity is regarded as the label of the entity. It iteratively learns the classifier via bootstrapping from both labeled data (seed entity alignments) and unlabeled data (predicated aligned entities). The embedding module adapts the triple score function of TransE $f_{\text{triple}}(\cdot)$ as defined in Eq. 2 by applying $f_{\text{triple}}(\cdot)$ on not only true triples from $\mathcal{G}_1$ and $\mathcal{G}_2$, but also all the "generated triples" obtained as follows: when an entity in a true triple, either head or tail, exists in the current set of aligned entities $\mathcal{S}$, replacing that entity by its aligned one in $\mathcal{S}$ generates a new triple. Note that $\mathcal{S}$ grows gradually with the iterations of bootstrapping. Specifically, the loss function for the embedding module is:

$$\mathcal{L}_e = \sum_{(h,r,t)\in\mathcal{T}_r} \max\left(0, \left[f_{\text{triple}}(h, r, t) - \gamma_1\right]\right) + \beta_1 \sum_{(h',r',t')\in\mathcal{T}'_r} \max\left(0, \left[\gamma_2 - f_{\text{triple}}\left(h', r', t'\right)\right]\right) \quad (13)$$

where $\mathcal{T}_r$ includes all the true triples in $\mathcal{G}_1$ and $\mathcal{G}_2$, as well as all the generated triples described above; $\mathcal{T}'_r$ contains all the corrupted triples generated by uniform negative sampling [4]. Note that this loss function is the sum of two parts in comparison to Eq. 3, which is called *limit-based loss function* proposed by Zhou et al. [83]; it minimizes both $f_{\text{triple}}(h, r, t)$ and $f_{\text{triple}}(h', r', t')$ by using two hyperparameters γ_1 and γ_2 to control them directly.

The alignment module of BootEA is a one-to-one classifier, which is different from the aligning method in Eq. 1 and uses a cross-entropy loss between the distribution of the entities in $\mathcal{G}_1$ versus the distribution of the predicted class (i.e., the aligned entity) from $\mathcal{G}_2$. All the pairs of entities in S are plugged into the following equation to compute the cross-entropy loss:

$$\mathcal{L}_a = -\sum_{e_1 \in \mathcal{E}_1} \sum_{e_2 \in \mathcal{E}_2} \phi_{e_1}(e_2) \log \pi(e_2 \mid e_1) \tag{14}$$

where $\phi_{e_1}(e_2)$ is a function that computes the labeling probability of e_1. If e_1 is labeled as e_2, the labeling distribution ϕ_{e_1} has all of its mass concentrated on e_2, i.e., $\phi_{e_1}(e_2) = 1$. If e_1 is unlabeled, ϕ_{e_1} is a uniform distribution; π is the classifier that predicts the aligned entity from $\mathcal{E}_2$ given $e_1 \in \mathcal{E}_1$.

The overall objective function of BootEA $\mathcal{L} = \mathcal{L}_e + \beta_2 \mathcal{L}_a$, where β_2 is a balancing hyperparameter.

NAEA [85] also formulates EA as a one-to-one classification problem but combines the translation-based and the GAT-based paradigms. Specifically, NAEA embeds *neighbor-level information* in addition to *relation-level information*. The neighbor-level information is embedded by aggregating the embeddings of the neighborhood with the attention mechanism as described in the GAT part of Sect. 4.2. Denote the neighbor-level representation of an entity e and the neighbor-level representation of a relation predicate r as $\mathrm{Ne}(e)$ and $\mathrm{Nr}(r)$, respectively. Then, based on the "translation" idea, the triple score function for the neighbor-level embedding is $f_{\text{triple}}(h, r, t) = \| \mathrm{Ne}(h) + \mathrm{Nr}(r) - \mathrm{Ne}(t) \|$. NAEA also uses the limit-based loss trick [83] and gets the following loss function for the neighbor-level embedding.

$$\begin{aligned}\mathcal{L}_1 = & \sum_{(h,r,t) \in \mathcal{T}_r} \sum_{(h',r',t') \in \mathcal{T}'_r} \\ & \times \max([f_{\text{triple}}(h, r, t) + \gamma_1 - f_{\text{triple}}\left(h', r', t'\right)], 0) \\ & + \beta_1 \sum_{(h,r,t) \in \mathcal{T}_r} \max\left(\left[f_{\text{triple}}(h, r, t) - \gamma_2\right], 0\right)\end{aligned} \tag{15}$$

The relation-level information is embedded using TransE. The overall loss function for the embedding module is $\mathcal{L}_e = \beta_2 \mathcal{L}_1 + (1 - \beta_2)\mathcal{L}_2$, where $\mathcal{L}_2$ is the same as Eq. 3 and β_2 is a hyperparameter.

The alignment module of NAEA is similar to BootEA, which uses a cross-entropy loss between the distribution of the entities in $\mathcal{G}_1$ and the distribution of the predicted class from $\mathcal{G}_2$ as follows.

$$\mathcal{L}_a = -\sum_{e_i \in \mathcal{E}_1} \sum_{e_j \in \mathcal{E}_2} \phi_{e_1}(e_2) \log \pi\left(e_j \mid e_i\right) \tag{16}$$

where $\phi_{e_1}(e_2)$ is the same as that in BootEA. The classifier $\pi\left(e_j \mid e_i\right)$ is defined as follows:

$$\begin{aligned}\pi\left(e_j \mid e_i\right) = & \beta_3 \quad \sigma\left(\mathrm{sim}\left(\mathrm{Ne}\left(e_i\right), \mathrm{Ne}\left(e_j\right)\right)\right) \\ & + (1 - \beta_3) \quad \sigma\left(\mathrm{sim}\left(\mathbf{e}_i, \mathbf{e}_j\right)\right)\end{aligned} \tag{17}$$

where $\mathrm{sim}(\cdot)$ is the cosine similarity and β_3 is a balancing hyperparameter.

TransEdge [47] addresses TransE's deficiency that its relation predicate embeddings are entity-independent, but in reality a relation predicate embedding should depend on its context, i.e., the head and tail entities. For example, the relation predicate `director` has different meanings in two different relation triples, (`Steve Jobs, director, Apple`) and (`James Cameron, director, Avatar`).

To address this issue, TransEdge proposes an *edge-centric translational embedding model* which regards the contextualized embedding of the relation predicate as the translation from the head entity to the tail entity. It contextualizes relation predicates as different edge embeddings, where the context of a relation predicate is specified by its head and tail entities. This is achieved by a triple score function as follows:

$$f_{\text{triple}}(h, r, t) = \| \boldsymbol{h} + \psi(\boldsymbol{h}_c, \boldsymbol{t}_c, \boldsymbol{r}) - \boldsymbol{t} \| \tag{18}$$

where $\psi(\boldsymbol{h}_c, \boldsymbol{t}_c, \boldsymbol{r})$ is the contextualized embeddings of a relation predicate, called the *edge embedding*.

The paper introduces two *interaction embeddings* $\boldsymbol{h}_c$ and $\boldsymbol{t}_c$ for encoding the head and tail entities' participation in the computation of the edge embeddings, respectively. The edge embeddings may be computed via two strategies, *context compression* and *context projection*. The first strategy, context compression, adopts multilayer perceptrons (MLPs) to compress the embeddings of the head entity, tail entity and the relation predicate as follows:

$$\psi(\boldsymbol{h}_c, \boldsymbol{t}_c, \boldsymbol{r}) = \mathrm{MLP}\left(\mathrm{MLP}([\boldsymbol{h}_c \| \boldsymbol{r}]) + \mathrm{MLP}([\boldsymbol{t}_c \| \boldsymbol{r}])\right) \tag{19}$$

The other strategy, context projection, projects the relation embedding onto the hyperplane of the head and the tail entities, and compute the edge embedding as:

$$\psi(\boldsymbol{h}_c, \boldsymbol{t}_c, \boldsymbol{r}) = \boldsymbol{r} - \boldsymbol{W}_{(h,t)}^{\top} \boldsymbol{r} \boldsymbol{W}_{(h,t)} \tag{20}$$

where $\boldsymbol{W} = \mathrm{MLP}([\boldsymbol{h}_c \| \boldsymbol{t}_c])$ is the normal vector of the hyperplane.

The alignment module of TransEdge uses a parameter sharing strategy to unify two different KGs, i.e., it forces a pair of aligned entities in the seed entity alignments to have the same embedding. TransEdge uses bootstrapping but newly aligned entities in each iteration are not processed with parameter sharing. To make these newly aligned entities close in the embedding space, a loss is added based on the embedding distance on the set $\mathcal{D}$ of newly aligned entities:

$$\mathcal{L} = \sum_{(e_1, e_2) \in \mathcal{D}} \| \boldsymbol{e}_1 - \boldsymbol{e}_2 \| \tag{21}$$

Other techniques that only use KG structure OTEA [38] adapts the *optimal transport* theory for EA. SEA [37] makes use of unlabeled data (unaligned entities) by adopting a cycle consistency restriction in the loss function. SX19 [43] models *multi-mappings* (i.e., many-to-many, one-to-many, or many-to-one) relations with a newly designed score function based on multiplication and complex vector space. AKE [28] first learns entity embeddings via TransE and then learns the unified vector space for $\mathcal{G}_1$ and $\mathcal{G}_2$ in an adversarial learning framework.

5.2 Techniques that exploit relation predicates and attributes

JAPE [45] makes use of attribute triples, albeit limited to only data types of attribute values (e.g., integers or strings), in addition to relation triples.

The embedding module of JAPE has two components: *structure embedding* and *attribute embedding*. The structure embeddings are obtained using TransE on $\mathcal{G}_1$ and $\mathcal{G}_2$ separately, producing two structure-based entity embedding matrices $\boldsymbol{E}_s^1$ and $\boldsymbol{E}_s^2$, respectively. The attribute embeddings are obtained by modeling the attribute co-occurrence within a same entity or across a pair of aligned seed entities. Specifically, a word embedding (*Skip-gram* [33] in their paper) is computed for every data type of attribute values based on the attribute co-occurrence as described above. Then the obtained word embedding for the data type is regarded as the embedding of the attribute itself. Then we can form an attribute-based entity embedding matrix consisting of the averaged attribute embeddings of all the entities, denoted as $\boldsymbol{E}_a^i$ for each KG, respectively, where $i = 1, 2$.

After obtaining both the structure and attribute-based entity embedding matrices, JAPE first computes cross-KG similarity $\boldsymbol{S}^{1,2}$ and inner-KG similarity for each KG (i.e., $\boldsymbol{S}^1$ and $\boldsymbol{S}^2$) based on the attribute-based entity embedding matrices:

$$\boldsymbol{S}^{1,2} = \boldsymbol{E}_a^1 \boldsymbol{E}_a^{2\top}; \quad \boldsymbol{S}^1 = \boldsymbol{E}_a^1 \boldsymbol{E}_a^{1\top}; \quad \boldsymbol{S}^2 = \boldsymbol{E}_a^2 \boldsymbol{E}_a^{2\top} \tag{22}$$

Then, it refines the embeddings by integrating the structural information with the following loss function.

$$\begin{aligned}\mathcal{L} = &\left\| \boldsymbol{E}_s^1 - \boldsymbol{S}^{1,2} \boldsymbol{E}_s^2 \right\|_F^2 \\ &+ \beta \left(\left\| \boldsymbol{E}_s^1 - \boldsymbol{S}^1 \boldsymbol{E}_s^1 \right\|_F^2 + \left\| \boldsymbol{E}_s^2 - \boldsymbol{S}^2 \boldsymbol{E}_s^2 \right\|_F^2 \right)\end{aligned} \tag{23}$$

where β is a hyperparameter that balances the importance of cross-KG similarity and inner-KG similarities.

KDCoE [8] builds on top of MTransE by shifting the entity embeddings by the embeddings of entity descriptions (i.e., literal descriptions for entities in KGs), which are treated as a type of special attribute triples where the attribute value is a literal description for the entity.

AttrE [51] is the first technique that makes use of attribute values. Moreover, it is the only EA technique that needs no seed alignments.

The embedding module of AttrE uses TransE to learn KG structure embeddings for the entities from $\mathcal{G}_1$ and $\mathcal{G}_2$. The main novelty of AttrE is to encode the semantics of the attribute values and three methods for encoding them are proposed: *averaged character embedding*, *aggregated character embedding by LSTM*, and *aggregated n-gram character embedding*. The aggregated n-gram character embedding has the best performance as reported in their paper, which uses the sum of n-grams of varying lengths to encode attribute values.

Another interesting idea proposed in AttrE, inspired by the "translation" idea in Eq. 2, is interpreting attribute triples (in addition to relation triples) as translating operation to learn the attribute embeddings as follows:

$$f_{\text{triple}}(e, a, v) = \|\boldsymbol{e} + \boldsymbol{a} - \tau(v)\| \tag{24}$$

where $\tau(v)$ is a function implementing one of the aforementioned encoding methods on the attribute value v. Thereby the same triple score function can be used to compute the plausibility of both relation and attribute triples uniformly. It helps shift the KG structure embeddings of $\mathcal{G}_1$ and $\mathcal{G}_2$ into the same vector space by minimizing the following loss function:

$$\mathcal{L}_s = \sum_{e \in \mathcal{E}_1 \cup \mathcal{E}_2} [1 - \text{sim}(\boldsymbol{e}_s, \boldsymbol{e}_c)] \tag{25}$$

where sim $(\boldsymbol{e}_s, \boldsymbol{e}_c)$ is the cosine similarity between the structure embedding $\boldsymbol{e}_s$ and the attribute embedding $\boldsymbol{e}_c$ of an entity e.

Besides making use of relation triples and attribute values, AttrE also aligns predicates (including both relation and attribute predicates) by exploiting the string similarity in the naming conventions of the predicates.

Finally, the inference module of AttrE predicts the aligned entity by computing the cosine similarity between the shifted structure embeddings.

MultiKE [80] uses multi-view learning on various kinds of features. The embedding module of MultiKE divides the features of KGs into three subsets called *views*: *name view*, *relation view*, and *attribute view*. Entity embeddings are learned for each view and then combined.

In the name view, an entity embedding is obtained from concatenating pre-trained word/character embeddings of the tokens in the entity name.

In the relation view, TransE is adopted to produce embeddings but with a logistic loss function:

$$\mathcal{L}_r = \sum_{(h,r,t)\in\mathcal{T}_r\cup\mathcal{T}_r'} \log\left(1+\exp\left(\zeta(h,r,t)\, f_{\text{triple}}\,(h,r,t)\right)\right) \tag{26}$$

where $\zeta(h,r,t)$ is 1 if (h,r,t) is a true triple, and -1 otherwise; $f_{\text{triple}}\,(h,r,t)$ is the same as in TransE.

In the attribute view, an attribute-value matrix $[\boldsymbol{a}\|\boldsymbol{v}]$ is first formed by the concatenation of the embeddings of attribute predicates and their values, and then the triple score function is defined as the head entity minus the result of a convolution neural network (CNN) on the attribute-value matrix, formally:

$$f_{\text{triple}}(e,a,v) = \|\boldsymbol{e} - \text{CNN}([\boldsymbol{a}\|\boldsymbol{v}])\| \tag{27}$$

This triple score function is then used to obtain the embeddings in the attribute view by minimizing the following objective function:

$$\mathcal{L}_a = \sum_{(e,a,v)\in\mathcal{T}_a} \log(1+\exp(f_{\text{triple}}(e,a,v))) \tag{28}$$

where $\mathcal{T}_a$ is a set of true attribute triples.

Next, the alignment module unifies the embedding spaces of the two KGs into the same vector space in each of the views. In the name view, the two KGs both use the same embedding scheme, i.e., pre-trained word embedding, so their embedding spaces are already unified. In the relation view and the attribute view, MultiKE performs the so-called *cross-KG entity/relation/attribute identity inference* to unify the embedding spaces as follows.

The entity identity inference is performed in both the relation and the attribute views. First, a strategy similar to BootEA [46] is adopted to generate triples as follows: when an entity in a true triple, either head or tail, exists in the current set of aligned entities $\mathcal{S}$, replace that entity by its aligned one in $\mathcal{S}$ generates a new triple. Then the sum of the plausibility (f_{triple}) of all the generated triples is minimized in both the relation and attribute views, which update all the embeddings. The updated embeddings are then fed into the relation and attribute identity inference below.

In the relation and attribute identity inference, first a similar strategy as AttrE [51] is adopted to derive soft relation and attribute predicate alignments by string similarity. Then the relation (attribute, respectively) identity inference generates triples in the relation view (attribute view, respectively) as follows: when a relation (attribute, respectively) predicate in a true triple exists in derived relation (attribute, respectively) alignments, replace the relation (attribute, respectively) predicate with its aligned counterpart. Then the sum of the plausibility (f_{triple}) of all the generated triples is minimized in both the relation and attribute views, which update all the embeddings.

The embeddings of an entity for the three views obtained above are combined into one embedding for the entity by averaging each view or minimizing a combination loss function. Finally, the inference module uses NNS based on the similarity between the combined entity embeddings.

COTSAE [73] alternatively trains structural and attribute embeddings and then combines the alignment results obtained from them.

6 GNN-based EA techniques

GNNs suit KGs' inherent graph structure so there are growing numbers of EA techniques based on GNNs recently. GNN-based EA techniques are categorized into GCN-based and GAT-based ones. They usually encode KG structure by the neighborhood of entities and many of them take attributes as input features for the embedding module because aligned entities tend to have similar neighborhood and attributes. Most GNN-based techniques use only seed entity alignments rather than other kinds of seed alignments in the training.

6.1 GCN-based EA techniques

GCN-Align [59] is the first study on GNN-based EA. Like many GNN-based EA techniques, GCN-Align learns entity embeddings from structural information of entities. GCN-Align also exploits attribute triples by treating them as relation triples. Specifically, GCN-Align uses two GCNs to embed the entities of $\mathcal{G}_1$ and $\mathcal{G}_2$ (one GCN for each KG) into a unified space with shared weight matrices, described by the following equation:

$$[\boldsymbol{H}_s^{(l+1)}\|\boldsymbol{H}_a^{(l+1)}] = \sigma(\hat{\boldsymbol{D}}^{-\frac{1}{2}}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-\frac{1}{2}}[\boldsymbol{H}_s^{(l)}\boldsymbol{W}_s^{(l)}\|\boldsymbol{H}_a^{(l)}\boldsymbol{W}_a^{(l)}]) \tag{29}$$

where $\boldsymbol{H}_s^{(l)}$ and $\boldsymbol{H}_a^{(l)}$ are the matrices for the structural and attribute embeddings, respectively; $\boldsymbol{W}_s^{(l)}$ and $\boldsymbol{W}_a^{(l)}$ are the weight matrices for these two types of embeddings, which are shared by the two GCNs. The matrix $\boldsymbol{H}^{(l)}$ in the vanilla GCN [24] (Equation 4) is replaced by a concatenation of structure and attribute embedding matrices. Unlike GCN, GCN-Align considers various types of relation predicates in KGs when computing the element $a_{ij} \in \boldsymbol{A}$. The new adjacency matrix $\boldsymbol{A}$ is designed as follows:

$$a_{ij} \in \boldsymbol{A} = \sum_{(e_i,r,e_j)\in\mathcal{T}_r} g_h(r) + \sum_{(e_j,r,e_i)\in\mathcal{T}_r} g_t(r) \tag{30}$$

where a_{ij} is the edge weight from the ith entity to the jth entity. Both (e_j,r,e_i) and (e_i,r,e_j) are triples in a KG. The functions $g_h(r)$ and $g_t(r)$ compute the number of head enti-

ties and the number of tail entities connected by relation r divided by the number of triples containing relation r, respectively. In this way, the adjacency matrix $\boldsymbol{A}$ helps model how the embedding information propagates across entities.

GCN-Align is trained by minimizing a margin-based loss function like Eq. 1. Taking into account of both structure and attribute embeddings, GCN-Align defines its alignment score function as follows:

$$f_{\text{align}}(e_1, e_2) = \beta \frac{\|\boldsymbol{h}_s(e_1) - \boldsymbol{h}_s(e_2))\|_{L_1}}{d_s} + (1-\beta)\frac{\|\boldsymbol{h}_a(e_1) - \boldsymbol{h}_a(e_2)\|_{L_1}}{d_a} \tag{31}$$

where $\boldsymbol{h}_s(\cdot)$ and $\boldsymbol{h}_a(\cdot)$ are the structure embedding with dimensionality d_s and attribute embedding with dimensionality d_a, respectively; β is used to balance the importance of these two embeddings.

HGCN [63] explicitly utilizes relation representation to improve the alignment process in EA. To incorporate the relation information, HGCN jointly learns entity and relation predicate embeddings in three stages as follows.

State 1 computes entity embeddings by a GCN variant named the *Highway-GCN* [40], which embeds entities into a unified vector space. The layer-wise highway gates control the forward propagation on top of the vanilla GCN layer, formulated as function T below:

$$T(\boldsymbol{H}^{(l)}) = \sigma\left(\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)} + \boldsymbol{b}^{(l)}\right) \tag{32}$$

$$\boldsymbol{H}^{(l+1)} = T\left(\boldsymbol{H}^{(l)}\right) \odot \boldsymbol{H}^{(l+1)} + \left(\boldsymbol{1} - T\left(\boldsymbol{H}^{(l)}\right)\right) \odot \boldsymbol{H}^{(l)} \tag{33}$$

where $\boldsymbol{H}^{(l)}$ is the output of the l^{th} layer and the input of the $(l+1)^{th}$ layer, $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are the weight matrix and bias vector, respectively; $\odot$ is element-wise multiplication. HGCN computes entity embeddings for both KGs separately and then maps the embeddings into a unified vector space using Eq. 1.

Stage 2 gets relation predicate embeddings based on their head and tail entity representations. This stage first computes the average embeddings of all the head entities and tail entities connected to the relation predicate, respectively. The two averaged embeddings are then concatenated as the embedding of the relation predicate after a linear transformation.

Stage 3 uses Highway-GCN again with the input being the concatenation of the entity embeddings computed in Stage 1 and the sum of all the relation predicate embeddings related to the entity. The alignment module maps the output of the Highway-GCN for the two KGs into a unified vector space with a loss similar to Eq. 1.

GMNN [69] formulates the EA problem as graph matching between two *topic entity graphs*. Every entity in a KG corresponds to a topic entity graph, which is formed by the one-hop neighbors of the entity and the corresponding relation predicates (i.e., edges). Such a graph represents the local context information of the entity. GMNN uses a graph matching model to model the similarity of two topic entity graphs, which indicates the probability of the two corresponding entities being aligned.

The graph matching model consists of four layers, including an input representation layer, a node-level matching layer, a graph-level matching layer, and a prediction layer. The input representation layer uses a GCN to encode two topic entity graphs and obtain entity embeddings. The node-level matching layer computes the cosine similarity between the embeddings of every pair of entities from two topic entity graphs. This layer further computes an attentive sum of entity embeddings as follows:

$$\overline{\boldsymbol{e}}_i = \frac{\sum_{j=1}^{|\mathcal{E}_2|} \alpha_{i,j} \cdot \boldsymbol{e}_j}{\sum_{j=1}^{|\mathcal{E}_2|} \alpha_{i,j}} \tag{34}$$

where $\alpha_{i,j}$ is the cosine similarity between entity e_i in a topic graph and entity e_j in another topic graph. This computation is done for entities from both two topic entity graphs. The resultant weighted sum of embeddings serves as the input to the graph-level matching layer. The graph-level matching layer runs a GCN on each topic entity graph to further propagate the local information throughout the topic entity graph. The output embeddings of the GCN is then fed to a fully connected neural network followed by the element-wise max and mean pooling method to get the graph matching representations for each topic entity graph. Finally the prediction layer takes the graph matching representation as input and uses a softmax regression function to predict entity alignment.

MuGNN [5] addresses the structural heterogeneity between KGs that may result in dissimilar embeddings of the entities that should be aligned. To reconcile the heterogeneity (i.e., the difference) between the structures of $\mathcal{G}_1$ and $\mathcal{G}_2$, MuGNN uses a multi-channel GNN in the embedding module to encode a KG in multiple channels toward *KG completion* and *pruning exclusive entities*.

One channel of MuGNN conducts KG completion by adding the relation predicates missing from a KG using the Horn rules for each KG, e.g., $marriedTo(x; y) \wedge liveIn(x; z) \Rightarrow liveIn(y; z)$, as extracted by AMIE+ [16]. The two resultant sets of rules are then transferred into each other via parameter sharing. The other channel of MuGNN prunes "exclusive entities," i.e., those entities that only appear in one of the two KGs.

Specifically, the multi-channel GNN is formulated as follows, assuming a two-channel MuGNN:

$$\text{MultiGNN}(\boldsymbol{H}^l; \boldsymbol{A}_1, \boldsymbol{A}_2) = \text{Pooling}(\boldsymbol{H}_1^{l+1}, \boldsymbol{H}_2^{l+1}) \tag{35}$$

$$\boldsymbol{H}_i^{l+1} = \mathrm{GCN}(\boldsymbol{A}_i, \boldsymbol{H}^l, \boldsymbol{W}_i), i = 1, 2 \tag{36}$$

where, similar to Equation 4, $\boldsymbol{H}^l$ is the input entity embeddings of the current layer while $\boldsymbol{H}_i^{l+1}$ is the output entity embeddings of this layer for the ith channel; $\boldsymbol{A}_i$ and $\boldsymbol{W}_i$ are the adjacency matrix and learnable weight matrix in the ith channel, respectively. At the end of the layer, pooling is used to combine the two channels. The adjacency matrices A_i are determined by different weighting schemes with self-attention and cross-KG attentions as follows.

$\boldsymbol{A}_1$ is determined based on self-attention, where element a_{ij} is the connectivity from e_i to e_j as follows:

$$a_{ij} = \mathrm{softmax}(c_{ij}) = \frac{\exp(c_{ij})}{\sum_{e_k \in N_{e_i} \cup \{e_i\}} \exp(c_{ik})} \tag{37}$$

Here, N_{e_i} is the neighborhood of e_i, and c_{ij} is the attention coefficient defined the same way as in Eq. 7.

$\boldsymbol{A}_2$ prunes exclusive entities by lowering the weight of the connectivity a_{ij} between those entities if one of them is an exclusive entity, formally:

$$a_{ij} \in \boldsymbol{A}_2 = \max_{r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2} \mathbf{1}((e_i, r_1, e_j) \in \mathcal{T}_1)\, \mathrm{sim}(r_1, r_2) \tag{38}$$

where $\mathcal{R}_1$ and $\mathcal{R}_2$ are the sets of relation predicates of $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. The function $\mathbf{1}(\cdot) = 1$ if $(e_i, r_1, e_j) \in \mathcal{T}_1$, and 0 otherwise. The function $\mathrm{sim}(r_1, r_2)$ is the inner-product similarity between two relation predicates r_1 and r_2.

To unify embeddings of $\mathcal{G}_1$ and $\mathcal{G}_2$ from the multi-channel GNN into a same vector space, the alignment module of MuGNN utilizes a variant of Eq. 1, which is the weighted sum of the seed entity alignments loss and the seed relation predicate alignments loss.

NMN [64] aims to tackle the structural heterogeneity between KGs. To address this issue, the technique learns both the KG structure information and the neighborhood difference so that the similarities between entities can be better captured in the presence of structural heterogeneity.

To learn the KG structure information, NMN's embedding module uses a GCN with highway gates to model the KG structure information with the input of a combination of $\mathcal{G}_1$ and $\mathcal{G}_2$ to be aligned. This module is pre-trained with a margin-based loss function (cf. Eq. 1) using seed entity alignments.

NMN then uses cross-graph matching to capture the neighborhood difference. A neighborhood sampling strategy is first used to select the more informative one-hop neighbors, based on the observation that the more often an entity and its neighbor appear in the same context, the more representative and informative the neighbor is for the entity. The cross-graph matching then compares the sampled neighborhood subgraph of an entity in the source KG with the subgraph of each candidate entity in the target KG to select an optimal aligned entity. A cross-graph vector is computed to indicate whether the entities are similar. The cross-graph matching is done by an attention mechanism.

NMN concatenate the entity embedding and its neighborhood representation to get the final embeddings for EA. EA is performed by measuring the Euclidean distance between entity embeddings.

CEA [78] considers the dependency of alignment decisions among entities, e.g., an entity is less likely to be an alignment target if it has already been aligned to some entity. The paper proposes a *collective EA* framework. It uses structural, semantic, and string signals to capture different aspects of the similarity between entities in the source and the target KGs, which are represented by three separate similarity matrices. Specifically, the structural similarity matrix is computed based on the embedding matrices via GCNs with cosine similarity, the semantic similarity matrix is computed from the word embeddings, and the string similarity matrix is computed by the Levenshtein distance between the entity names. The three matrices are further combined into a fused matrix. CEA then formulates EA as a classical stable matching problem on the fused matrix to capture interdependent EA decisions, which is solved by the *deferred acceptance* algorithm [42].

Other GCN-based EA techniques RDGCN [62], which is similar to HGCN, utilizes relation information and extends GCNs with highway gates to capture the neighborhood structural information. RDGCN differs from HGCN in that it incorporates relation information by the attentive interaction. AVR-GCN [76] considers multi-mappings under the GCN paradigm and learns the embeddings of entities and relation predicates simultaneously for KGs. Specifically, it first learns these embeddings via an embedding model named *vectorized relational GCN* and then uses a weight sharing mechanism to join (e.g., via concatenation or vector transformation) those embeddings into a unified vector space. HMAN [72] takes into account even more other types of information such as relation predicates, attribute values, and entity descriptions besides the structural information. Specifically, HMAN employs a pre-trained BERT model [10] to capture the semantic relatedness of the descriptions of two entities that cannot be measured directly. SSP [35] uses both translation- and GNN-based paradigms. It captures local semantics from relation predicates and global structural information by a structure and semantics preserving network. CG-MuAlign [86] addresses structural heterogeneity by collectively aligning entities via the attention mechanism. XS20 [70] is another EA technique that addresses the many-to-one alignment problem in its inference module. It models EA as

a *task assignment* problem and solves it by the *Hungarian algorithm* [25].

6.2 GAT-based EA Techniques

KECG [27] aims to reconcile the issue of structural heterogeneity between KGs by jointly training both a GAT-based *cross-graph model* and a TransE-based *knowledge embedding model*.

The cross-graph model in KECG embeds entities with two GATs on the two KGs, which encode the graph structure information. The attention mechanism in the GATs helps ignore unimportant neighbors and mitigate the issue of structural heterogeneity. The GATs' projection matrices $\boldsymbol{W}$ (cf. Eq. 5) are set to diagonal matrices, which reduces the number of parameters to be learned and increases the model generalizability.

As usual, KECG uses attention mechanisms as described in Sect. 4.2 and margin-based loss for the cross-graph model as described in Sect. 3.

The knowledge embedding model in KECG uses TransE to encode the structural information in each KG separately. The overall objective function of KECG is a weighted sum of the loss functions from the cross-graph model and the knowledge embedding model.

AliNet [48] is based on the observation that some aligned entities from $\mathcal{G}_1$ and $\mathcal{G}_2$ do not share similar neighborhood structures. Such aligned entities may be missed by the other GNN-based EA techniques, because they rely on similar neighborhood structures for EA. AliNet addresses the issue by considering both direct and distant neighbors.

AliNet learns entity embeddings by a controlled aggregation of entity neighborhood information. Without loss of generality, we describe the process for two-hop neighborhood below, although any number of hops is applicable. First, a GCN is used to aggregate the direct (i.e., one-hop) neighbors' information. Let the embedding of an entity e_i at the lth layer be $\boldsymbol{e}_{i,1}^{(l)}$ after one-hop neighbor aggregation. Then for two-hop neighbors, an attention mechanism is used to indicate their contribution to the embedding of e_i as follows:

$$\boldsymbol{e}_{i,2}^{(l)} = \sigma\Big(\sum_{j\in\mathcal{N}_2(i)\cup\{i\}} \alpha_{ij}^{(l)} \boldsymbol{W}_2^{(l)} \boldsymbol{e}_j^{(l-1)}\Big) \tag{39}$$

where $\boldsymbol{e}_j^{(l-1)}$ is the embedding of e_j at the $(l-1)$th layer of the GCN; $\mathcal{N}_2(\cdot)$ is the set of the two-hop neighbors of e_i; $\boldsymbol{W}_2^{(l)}$ is a learnable weight matrix. To retain the difference between e_i and its neighbors, the attention coefficient $c_{ij}^{(l)}$ is computed using two different transformation matrices $\boldsymbol{M}_1^{(l)}$ and $\boldsymbol{M}_2^{(l)}$ for $e_i^{(l)}$ and $e_j^{(l)}$, respectively:

$$c_{ij}^{(l)} = \textsc{LeakyReLU}\left[\left(\boldsymbol{M}_1^{(l)}\boldsymbol{e}_i^{(l)}\right)^\top \left(\boldsymbol{M}_2^{(l)}\boldsymbol{e}_j^{(l)}\right)\right] \tag{40}$$

At the end of each layer of AliNet, the information from one-hop and two-hop neighbors is combined with a gating mechanism, i.e., the embedding of entity e_i at the lth layer is computed as follows:

$$\boldsymbol{e}_i^{(l)} = g\left(\boldsymbol{e}_{i,2}^{(l)}\right)\cdot \boldsymbol{e}_{i,1}^{(l)} + \left(1 - g\left(\boldsymbol{e}_{i,2}^{(l)}\right)\right)\cdot \boldsymbol{e}_{i,2}^{(l)} \tag{41}$$

where $g(\cdot)$ is the gate, $g(\boldsymbol{e}_{i,2}^{(l)}) = \sigma\left(\boldsymbol{M}\boldsymbol{e}_{i,2}^{(l)} + \boldsymbol{b}\right)$, and $\boldsymbol{M}$ and $\boldsymbol{b}$ are the weight matrix and the bias.

MRAEA [32] considers meta-relation semantics including relation predicates, relation direction, and inverse relation predicates, in addition to structural information learned from the structure of relation triples. The meta-relation semantics are integrated into structural embedding via *meta-relation-aware embedding* and *relation-aware GAT*.

To compute the meta-relation-aware embeddings (concatenation of entity and relation predicate embeddings), we first extend the set of relation triples by creating an "inverse triple" for each triple by replacing the original relation predicate with an "inverse relation predicate" while keeping the same head and tail entities unchanged. Second, the entity and relation embedding components of the meta-relation-aware embeddings of the target entity are computed by averaging those of the neighbor entities, respectively.

The relation-aware GAT generates a structure-and-relation -aware embedding of each entity by attending the meta-relation-aware embeddings of the target entity's neighbors. Specifically, the GAT's attention coefficient c_{ij}, which indicates the importance of both the neighbor entity e_j and the connecting relation predicate r_k to the target entity e_i, is computed as:

$$c_{ij} = \boldsymbol{w}^T\left[\boldsymbol{e}_i \| \boldsymbol{e}_j \| \frac{1}{|\mathcal{M}_{ij}|}\sum_{r_k\in\mathcal{M}_{ij}} \boldsymbol{r}_k\right] \tag{42}$$

where the embeddings $\boldsymbol{e}_i$, $\boldsymbol{e}_j$, and $\boldsymbol{r}_k$ are obtained from the meta-relation-aware embeddings; $\boldsymbol{w}$ is a learnable weight vector; $\mathcal{M}_{ij} = \{r_k \mid (e_i, r_k, e_j) \in \mathcal{T}\}$ is the set of relation predicates that link from e_i to e_j, which incorporates relation features into the attention mechanism.

As usual, MRAEA is trained with a margin-based loss function like Eq. 1.

EPEA [60] learns embeddings of entity pairs via a *pairwise connectivity graph* (PCG) rather than embeddings for individual entities. EPEA first generates the PCG, whose nodes are pairs of entities from $\mathcal{G}_1$ and $\mathcal{G}_2$. Given two entity pairs $(e_{1,i}, e_{2,i})$ and $(e_{1,j}, e_{2,j})$ in the PCG, an edge is added between the two entity pairs if there is a relation predicate r_1 connecting $e_{1,i}$ to $e_{1,j}$ in $\mathcal{G}_1$ and a relation predicate r_2 connecting $e_{2,i}$ to $e_{2,j}$ in $\mathcal{G}_2$. After generating the entity pairs

as the nodes of the PCG, EPEA uses a CNN to encode the attributes of entity pairs into embeddings based on attribute similarity. These attribute embeddings are then fed into a GAT that further incorporates structural information and produce a score, which indicates the probability of a pair consisting of aligned entities. This scoring function is then used as f_{align} in Eq. 1 to train the whole model. The inference module predicts aligned entities by performing binary classification on the scoring function value with the input being the embeddings of entity pairs.

AttrGNN [31] learns embeddings from both relation triples and attribute triples in a unified network. It partitions each KG into four subgraphs containing attribute triples of entity names, attribute triples of literal values, attribute triples of digital values, and the remaining triples (i.e., relation triples), respectively. For each subgraph, entity embedding is computed based on attributes as well as the KG structure using GAT; then a similarity matrix between $\mathcal{G}_1$ and $\mathcal{G}_2$ is computed based on the entity embedding. Finally, the four similarity matrices are averaged to yield a final similarity matrix for the inference module.

7 Datasets and experimental studies

We discuss the limitations of existing datasets and experimental studies, present our proposed datasets addressing the limitations, and report on a comprehensive experimental study on representative EA techniques using our datasets.

7.1 Limitations of existing datasets

There are several significant limitations of existing datasets, namely *bijection*, *lack of name variety* and, *small scale*, which are detailed below.

Bijection Many existing papers, including some of the benchmarking papers, have used datasets that consist of two KGs where almost every entity in one KG has one and only one aligned entity in the other KG, i.e., there is bijection between the two KGs. Such datasets have been generated from different language versions of Wikipedia (e.g., DBP15K [45] and $\text{SRPRS}_{\text{multi}}$ [18]) and the application argued for such datasets is aligning two KGs in different languages, i.e., *multilingual EA*. However, such application instances are infrequent in real life.

We argue that the following scenario is more common: two KGs come from different sources, e.g., a generic KG built from Wikipedia and the other from a domain-specific source such as medicine, locations, flights and music. The difference in the sources is typically not language but the coverage of knowledge, so the two KGs are complementary to each other and aligning them helps enrich them. Therefore, *non-bijection* between the KGs is desired. A recent paper [82] also points out that bijection is an unrealistic setting and created *DBP-FB*, a dataset consisting of two KGs built from different sources, DBpedia and Freebase. It is a great step toward non-bijection datasets, but unfortunately, a big limitation of DBP-FB is that it does not contain any generic attribute (e.g., year, address, etc.) triples except entity names and hence does not suffice the need of most recent EA techniques, which make heavy use of generic attributes as features. From our experiments (Table 5) we see that most recent EA techniques use generic attributes as input features which are essential for effectiveness. Creating datasets coming from different sources is challenging. A recent industrial benchmark dataset MED-BBK-9K [81] is built from different sources. However, this dataset also has the bijection problem and the size is small: the number of unique entities covered in this dataset is less than 10, 000.

Lack of name variety Unlike recently proposed datasets MED-BBK-9K and DBP-FB, most previous EA datasets are constructed from KGs with the same source. For example, DWY100K [46] and its resampled version $\text{SRPRS}_{\text{mono}}$ [18] consist of KG pairs (DBpedia-Yago and DBpedia-Wikidata) with the same primary source, Wikipedia. Thus, the names of entities from two KGs may have the same surface label; such names become "tricky features," w<>hich can be used to achieve 100% accurate EA easily (we call this the *lack of name variety problem*). To address this problem, Sun et al. [49] remove all the entity names from DWY100K. However, this is an overkill because, in real-world settings, KGs do contain entity names as attributes but just have variety in the names of the same real-world entity. The recently proposed dataset DBP-FB have significant portion of entities (42%) with different entity names and hence have good name variety due to the different data sources DBpedia and Freebase, but its lack of generic attribute triples limits its use as mentioned earlier. We need datasets that have significant amount of generic attribute triples and variety in the names. Our proposed datasets address these issues.

Small scale Most existing datasets are of small (e.g., MED-BBK-9K contains 9162 unique entities) to medium (e.g., DBP-FB contains 29,861 unique entities) sizes. Our proposed datasets contain up to 600,000 unique entities from the two KGs combined.

Table 3 summarizes the key properties of various datasets. Our proposed benchmark DWY-NB has all the three desirable properties: non-bijection, name variety, and large size. Some studies consider the property of whether the dataset is multilingual or monolingual. We do not view it as an essential property that affects the utility of the datasets, since EA techniques that utilize semantic information of KGs such as attributes can first translate it into the target language. Note that all the current multilingual datasets have the bijection problem discussed earlier.

Table 3 Dataset/Benchmark comparisons

Name	Non-Bijection	Name variety	Size*	Language
DBP15K [45]	No	No	Small	Multilingual
DWY100K [46]	Large	No	No	Monolingual
$\text{SRPRS}_{\text{multi}}$ [18]	No	No	Small	Multilingual
$\text{SRPRS}_{\text{mono}}$ [18]	No	No	Small	Monolingual
DBP-FB [82]]	Yes	Yes	Medium	Monolingual
DBP-FB [82]]	Yes	Yes	Medium	Monolingual
MED-BBK-9K [81]	No	Yes	Small	Monolingual
DWY-NB (Our Proposed)	Yes	Yes	Large	Monolingual

*Dataset size represent the number of unique entities in the dataset: **Small** (< 20,000); **Medium** (20,000–50,000); **Large** (> 50,000)

7.2 Limitations of existing experimental studies

Several recent studies aim at benchmarking EA techniques. Sun et al. [49] re-implemented 12 representative EA techniques, but the re-implementation missed important components in some techniques such as predicate alignments in AttrE [51] and MuGNN [5]). To avoid such problems, we use the original code of each compared technique. Another limitation of Sun et al. [49] is that they only used bijection datasets. The study by Zhao et al. [82] does not include experiments on techniques that use attribute triples such as AttrE [51] and MultiKE [80], but as shown in our experiments, most recent techniques use attribute triples and they have much best performance. Zhang et al. [81] proposes a new dataset MED-BBK-9K, but it has the limitations of bijection and small size, making the study less comprehensive.

7.3 Our proposed benchmark DWY-NB

To address the limitations of existing datasets, we propose a new benchmark called **DWY-NB** where NB stands for non-bijection. This benchmark consists of two regular-scale datasets and large-scale ones described at the end of this subsection; each of the regular-scale datasets consists of a pair of KGs that can be used for the evaluation of EA techniques. We call the two datasets **DW-NB** and **DY-NB**. The two KGs of DW-NB are subsets of DBpedia and Wikidata [56], respectively. The two KGs of DY-NB are subsets of DBpedia [1] and Yago [19], respectively. We choose these sources as starting points because they contain rich relation and attribute triples.

Now we explain how the datasets are generated. For ease of explanation, we next use DW-NB as an example while the process for DY-NB is similar. We start from the list of aligned entities between two KGs (*DBpedia and Wikidata*) from the dataset DWY100K from [49], we call this list the *seed entity alignments*. This seed entity alignment contains a list of 100,000 alignments between the entities in DBpedia and Wikidata, which originally provided by the DBpedia website.[2] We extract all those triples from DBpedia (and Wikidata, respectively) that contain the entities listed in the seed entity alignments to form a sub-graph of DBpedia (and Wikidata, respectively), and this sub-graph of DBpedia and sub-graph of Wikidata become the pair of KGs in our DW-NB dataset.

To address the bijection problem, we randomly remove a certain percentage (25% by default) of the entities from each of the two KGs; we make sure the entities removed from one KG is different from the entities removed from the other KG so that not every entity in one KG will have an aligned entity in the other KG. As a result, by default 50% of the entities in the two KGs combined do not have aligned ones. We also vary the proportion of aligned entities in our benchmark.

To address the lack of name variety problem, we add variety to the entity name attributes as follows. In every KG source (DBpedia, Wikidata, or Yago), there are often multiple attributes corresponding to the name of the entity, which we refer to as *name attributes*. These multiple name attributes have different attribute values (i.e., entity names) as a result of the curation process of the KGs conducted by different humans. For an entity with multiple name attributes, we select a name attribute that is different from that of the corresponding entity in the other KG, if any. Thus, the above procedure provides variety in entity names between the KGs in our datasets. After this process, it turns out that 36% of the aligned entities have different entity names. The statistics of DWY-NB are listed in Table 4. To conduct scalability experiment in Experiment 5 of Section 7.4, we further generate larger versions (100K, 300K, and 600K entities in each KG of the KG pair) of the dataset DW-NB in the same way as described above. Details of all the datasets can be found at (https://github.com/ruizhang-ai/EA_for_KG).

[2] https://wiki.dbpedia.org/downloads-2016-10.

Table 4 Statistics of our proposed benchmark DWY-NB

Subset	Unique entities	Aligned entities	Predicates	Relationship triples	Attribute triples
DW-NB					
DBpedia	84,911	50,000	545	203,502	221,591
Wikidata	86,116		703	198,797	223,232
DY-NB					
DBpedia	58,858	15,000	211	87,676	173,520
Yago	60,228		91	66,546	186,328

7.4 Experiments and results

We conduct five sets of experiments. *Experiment 1*: Following the literature [7,46,64], the main evaluation measure for the effectiveness of EA techniques is hits@1 (or hits@k) which indicates the percentage of entities that have the correct aligned entity in the top-k predicated aligned entities. *Experiment 2*: We evaluate the effect of attribute triples on the effectiveness of EA techniques as using attribute triples has been a trend of recently proposed EA techniques. *Experiment 3*: In addition to Experiment 1, we also evaluate the effectiveness of EA techniques via direct downstream applications. *Experiment 4*: At the end of Sect. 3, we argue that whether a technique is designed to conduct multilingual EA is not an essential characteristic because we can perform an automatic translation on the semantic information into the target language so that both KGs are in the same language. This experiment justifies this argument. *Experiment 5*: We investigate how various techniques scale up with dataset sizes.

Compared techniques We compare representative techniques that provide access to their code. We do not make any changes to the code. We use the parameter settings suggested in the original papers for each technique. For detailed parameter settings of each technique, readers may refer to the corresponding papers.

Datasets We use the two datasets in DWY-NB (cf. Sect. 7.3) for the experiments. Note that many EA techniques such as Cao et al. [5]; Ye et al. [76] use manually created seed attribute/relation predicate alignments which positively impacts the performance, while some other techniques do not. To be able to isolate the effect of the factor being evaluated for a certain experiment in our experimental study (e.g., the effect of seed entity alignments proportions, the effect of attribute triples, etc.), we have aligned the predicates between the KGs in those experiments so that predicate alignments have the same effect on the performance of all the techniques no matter whether they take measures to align predicates. If we did not align the predicates in our data, then the techniques that do not take measures to align predicates might have poorer performance due to unaligned predicates rather than due to the effect of the factor being evaluated.

Environments We run the experiments with an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz processor, 128GB main memory, a Nvidia Tesla GPU with 32GB memory, and Ubuntu 20.4. The programming language and libraries include Python, TensorFlow, Torch, etc. depending on the language used for the original code.

Experiment 1: the effect of seed entity alignments This experiment evaluates the accuracy of EA in terms of Hits@k while varying the amount of seed entity alignments used for training from 10 to 50% of the total available set of seed entity alignments (50,000 for DW-NB and 7500 for DY-NB). Higher hits@k means better accuracy. Table 5 shows the result. The accuracy of all the techniques gets higher with more seeds, which is expected since more seeds provide more supervision.

Overall, AttrE and MultiKE have much better performan ce than the others especially when less seed entity alignmen ts are available. This is because they make great use of vari ous types of features such as attributes and relation predicat es while the other techniques do not. AttrE's performance does not change while varying the amount of seeds since it does not use seed alignments, so when seed alignments are hardly available, AttrE is the clear winner. Only when the amount of seed entity alignments reaches 50%, NMN has slightly higher Hits@1 than AttrE. In general, to achieve good alignment results, supervised models require seed alig nments with the proportion of at least 30%.

Among the GNN-based techniques, RDGCN and NMN are the top-2 in terms of Hits@1. It is worth noting that the top-2 from both translation- and GNN-based techniques exploit attribute triples, and we can see that on average, the techniques that exploit attribute triples achieve much better performance than the techniques that do not. JAPE has poorer performance compared to other techniques that use attributes because it uses very limited information from attributes, only the data type of attribute value.

Experiment 2: the effect of attribute triples This experiment evaluates how much benefit may be gained by exploiting attribute triples. For every EA technique, we compare the performance of a "using-attribute" version versus a "not-using-attribute" version as follows. For a technique

Table 5 Experiment 1: The effect of the amount of seed entity alignments on EA accuracy in terms of Hits@k (%)

Technique		Seed: 10%		Seed: 20%		Seed: 30%		Seed: 40%		Seed: 50%	
		Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10
DW-NB											
Translation-based	MTransE	2.82	10.45	5.42	18.72	7.88	25.75	10.42	31.44	12.98	36.00
	IPTransE	5.98	13.45	7.54	18.78	12.90	24.61	16.32	32.86	23.54	35.98
	BootEA	8.12	16.15	12.54	20.13	17.92	28.38	21.46	35.16	25.44	37.57
	TransEdge	22.98	48.12	38.29	56.22	45.27	68.95	49.26	75.25	54.85	79.68
	JAPE	4.62	7.87	8.62	14.43	12.57	19.96	17.20	27.32	19.91	30.63
	MultiKE	80.25	87.58	82.56	88.92	84.06	90.05	84.87	91.24	85.21	95.06
	AttrE	**87.98**	**95.80**	**87.98**	**95.80**	**87.98**	**95.80**	**87.98**	**95.80**	87.98	**95.80**
GNN-based	MuGNN	13.49	37.79	20.96	49.28	26.92	56.77	31.09	61.43	34.41	64.96
	AliNet	14.58	31.46	18.55	35.84	24.34	50.46	28.39	55.46	35.31	58.22
	KECG	18.95	34.17	24.32	40.78	30.24	48.66	35.29	52.12	39.40	62.31
	GCN-Align	12.40	30.18	20.04	41.56	24.76	48.52	29.02	53.43	31.80	56.20
	HGCN	58.08	62.15	63.14	68.15	78.97	86.51	84.25	90.75	88.54	91.54
	GMNN	71.32	74.24	75.34	79.23	80.98	82.23	82.67	85.87	84.59	88.64
	RDGCN	59.22	62.98	64.22	68.98	79.02	87.12	85.34	90.45	88.21	93.23
	CEA	50.13	52.31	63.25	64.12	80.32	84.21	84.34	85.54	86.58	88.34
	MRAEA	53.75	54.74	64.58	66.12	81.54	85.97	83.54	86.02	84.06	87.55
	NMN	51.45	59.78	68.21	72.54	84.03	88.21	85.65	90.54	**88.69**	95.46
DY-NB											
Translation-based	MTransE	0.01	0.15	0.01	0.39	0.08	0.68	0.08	1.39	0.13	1.89
	IPTransE	1.54	9.87	5.67	25.76	14.55	36.45	15.77	45.81	17.33	52.18
	BootEA	2.15	14.19	8.47	38.15	15.77	48.32	17.22	57.15	19.24	58.14
	TransEdge	22.98	47.50	37.85	64.85	48.98	72.15	58.95	76.54	62.49	78.54
	JAPE	0.70	1.83	1.57	3.37	1.40	3.27	1.37	1.77	2.37	4.97
	MultiKE	81.87	88.05	82.11	89.26	84.97	90.84	87.22	92.05	89.25	93.58
	AttrE	**90.44**	**94.23**	**90.44**	**94.23**	**90.44**	**94.23**	**90.44**	**94.23**	90.44	94.23
GNN-based	MuGNN	19.16	51.41	27.40	62.69	31.60	68.56	34.73	71.24	37.15	74.07
	AliNet	13.54	28.53	14.25	31.69	25.39	58.31	28.98	56.12	34.59	64.12
	KECG	11.19	23.65	14.89	27.25	20.95	34.48	22.81	35.44	24.71	37.15
	GCN-Align	8.56	25.09	17.88	43.88	24.36	53.43	31.29	62.44	33.56	67.88
	HGCN	52.54	64.51	65.87	77.40	71.14	85.64	71.45	85.64	74.54	87.48
	GMNN	62.34	70.34	64.32	67.34	75.57	77.47	78.65	82.65	82.34	85.62
	RDGCN	53.13	65.30	67.28	78.21	74.54	85.22	77.45	87.43	78.67	89.45
	CEA	55.24	58.97	64.35	65.42	74.56	78.42	77.78	80.95	78.91	83.24
	MRAEA	52.46	53.20	60.33	64.54	73.71	78.52	74.25	78.66	76.22	80.15
	NMN	55.74	64.78	62.54	70.54	75.87	80.54	84.55	88.69	**90.78**	**94.77**

*Techniques that use attribute triples are underlined; boldfaced numbers indicate the best performance in that experiment. The rest tables and figures follow this convention

*AttrE does not use any seed alignments

that exploits attribute triples by design, we get the performance of its "not-using-attribute" version by only using relation triples (and not using attribute triples) to compute the entity embeddings. For a technique that only uses relation triples by design, we get the performance of its "using-attribute" version by applying a naive way of exploiting attribute triples, i.e., treating the attribute triples as relation triples which means treating the attribute values as nodes in the graph. Figure 3 shows the results. The proportion of seed entity alignments used in this experiment is 30% and the results on other proportions have a similar behavior. For every technique, the "using-attribute" version outperfor

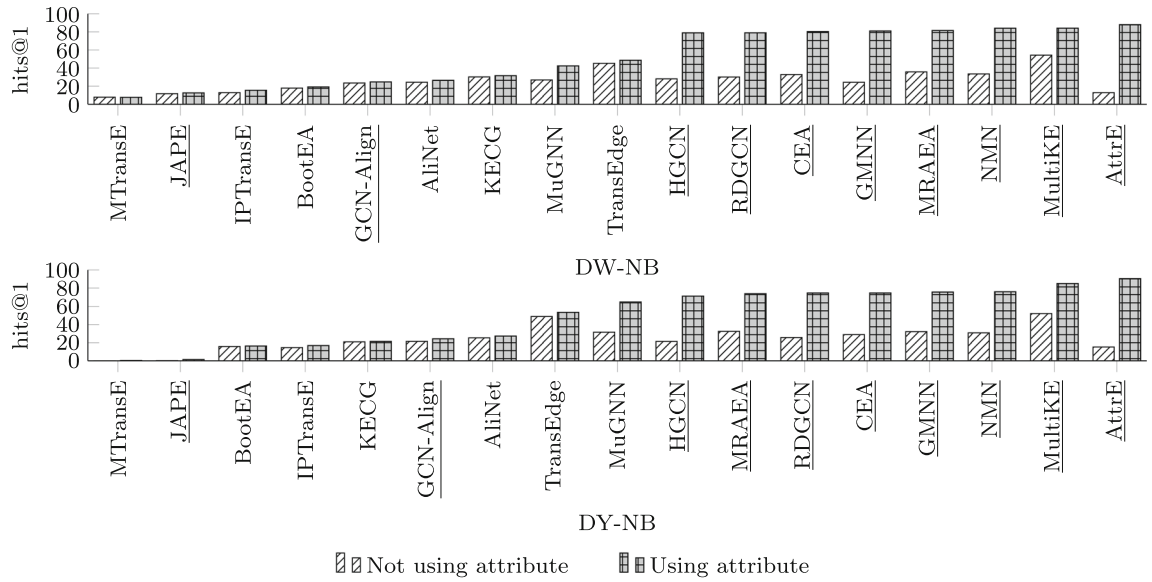

Fig. 3 Experiment 2: Using attributes versus Not using attributes (sorted by performance of "using-attribute" version)

ms the "not-using-attribute" version, especially for those tec hniques that use attribute triples by design. These show that making good use of attribute triples can improve the accura cy significantly. Among them, the gap between the two versions of AttrE is huge, because AttrE does not use seed alignments and heavily rely on attributes to train the alignment module. MultiKE uses both seed alignments and attribute triples to produce the unified embedding space and hence has relatively smaller gap between the two versions, but still using attributes provides substantial gains. When the KGs do not contain attribute triples but seed align ments are available, MultiKE is the winner. In comparison, the performance of GNN-based techniques (e.g., HGCN, RDGCN, NMN) drops significantly (most by 50%) when "not-using-attribute," as they heavily rely on entity names to initialize the node embeddings in the embedding module. In the absence of entity names, node embeddings are randomly initialized which leads to poor performance. Interestingly, the techniques that do not use attribute triples by design also gets better performance with the "using-attribute" version, even this is by the naive way of treating attribute triples as relation triples.

As a case study to understand the benefit of attribute triples intuitively, we examine the following example: `dbp:Ali_L ohan` and `dbp:Lindsay_Lohan` are siblings that have the same neighbors: `dbp:Michael_Lohan` and `dbp:Di na_lohan`, which represent the father and mother, respectively. EA techniques that only use the graph structure information cannot distinguish `dbp:Ali_Lohan` from `dbp:Lindsay_Lohan`, which may lead to a misalignment. EA techniques that exploit attribute triples can use the attribute triples, such as `(dbp:Lindsay_Lohan, birth_date, 1986-07-02)` and `(dbp:Ali_Loh an, birth_date, 1993-12-22)` to distinguish them. Such cases are very common in the two KGs.

Experiment 3: the effect of the alignment module on KG embeddings The training in EA techniques optimize for two objectives, the *KG embeddings* and the *alignment of two KGs* (either jointly or alternatively), rather than merely the KG embeddings, so it might not produce the best KG embeddings. This experiment evaluates how the quality of the KG embeddings obtained from EA techniques are affected compared to the KG embeddings obtained from pure KG embedding techniques (TransE for translation-based and GCN for GNN-based techniques) via downstream applications of KGs. Following previous studies in EA techniques [49,82], we use two common downstream tasks for this purpose: *link prediction* for translation-based techniques and *node classification* for GNN-based techniques, detailed as follows. The link prediction task aims to predict t given h and r of a relation triple. Specifically, first a relation triple is corrupted by replacing its tail entity with all the entities in the dataset. Then, the corrupted triples are ranked in ascending order by the plausibility score computed as $\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}$. Since true triples (i.e., the triples in a KG) are expected to have smaller plausibility scores and rank higher in the list than the corrupted ones, hits@10 (whether the true triples are in the top-10) is used as the measure for the link prediction task. The node classification task aims to classify nodes and determine their labels. Given the embedding of a node, a simple classifier SVM [9] with twofold cross-validation is trained to predict the entity type (e.g., `person, organization`, etc.) of the node. Table 6 shows the accuracy of downstream

Table 6 Experiment 3: effects on downstream tasks

Technique	DBP-WD (seed)			DBP-YAGO (seed)		
	(10%)	(30%)	(50%)	(10%)	(30%)	(50%)
Link Prediction (Evaluating Translation-based Models)						
MultiKE	**88.76**	**88.98**	**89.52**	98.62	**98.87**	98.07
AttrE	88.50	88.50	88.50	**98.75**	98.75	**98.75**
TransE*	87.45	87.45	87.45	98.42	98.42	98.42
TransEdge	85.27	85.71	86.40	93.24	93.54	93.76
JAPE	83.24	83.71	83.09	75.03	75.32	75.66
IPTransE	81.06	81.23	81.78	93.10	93.55	93.91
BootEA	80.41	80.90	81.66	94.11	94.54	94.85
MTransE	80.10	80.33	80.69	93.81	94.31	94.74
Node Classification (Evaluating GNN-based Models)						
GCN*	**64.93**	**64.93**	**64.93**	**68.21**	**68.21**	**68.21**
NMN	62.25	62.74	62.85	66.46	66.57	66.79
CEA	60.06	60.24	60.39	65.95	66.31	66.66
MRAEA	57.95	58.34	58.56	65.77	65.87	66.37
GCN-Align	54.05	54.52	54.93	61.54	62.03	62.37
HGCN	53.93	54.11	54.59	65.79	66.24	66.48
GMNN	52.21	52.36	52.67	67.63	67.87	67.95
MuGNN	51.63	51.96	52.46	43.16	43.45	43.58
RDGCN	51.31	51.46	51.86	56.80	57.00	57.28
KECG	44.83	45.19	45.50	57.75	58.12	58.45
Alinet	42.68	42.98	43.16	37.96	38.45	38.72

*Baseline

applications on DWY-NB with 10%, 30%, and 50% of seed entity alignments. The accuracy increases with the amount of seed alignments but not significantly.

Translation-based EA techniques are compared against TransE, a pure KG embedding technique. MultiKE and AttrE have higher link prediction accuracy than TransE whi le the others do not. This is because MultiKE and AttrE make great use of various types of information including attribute triples as input features, which improve the quality of KG embeddings.

GNN-based EA techniques are compared against GCN, a pure KG embedding technique. All of the GNN-based techniques have lower node classification accuracy than GC N; the best one is NMN (about 2% lower than GCN). The techniques that use attribute triples achieve better accuracy than those that do not.

In summary, the KG embeddings obtained from EA tech niques may have slightly better or worse performance in do wnstream tasks depending on the paradigm of KG structure embeddings, details provided in previous paragraphs.

Experiment 4: Multilinguality This experiment evaluates how various techniques perform on multilingual KGs with the approach of first translating into the same language. Following previous studies [82] we use the multilingual dataset $\text{SRPRS}_{\text{multi}}$ [18], which contains two KG pairs EN-DE and EN-FR. To translate the attribute triples into English, we use a popular open-source translation tool *Fairseq* [36]. For each technique we run a version "not using attributes" (the original techniques) and a version "using attributes" (the translation approach). The results are shown in Table 7. All the techniques have significant improvement by using attributes via the translation approach, including the techniques that can perform multilingual EA by design (mostly all the GNN-based techniques). AttrE and MultiKE are not designed for multilingual EA, but via the translation approach both have comparable performance to the techniques designed for multilingual EA. These validate our argument that techniques designed for m onolingual EA can perform multilingual EA well by exploit ing semantic information (such as attributes) and automatic translation.

Experiment 5: Scalability This experiment evaluates how various techniques perform as data sizes grow. We use the same way described in Sect. 7.3 to create EA datasets with varying numbers of entities 100K, 300K, and 600K in each KG of the KG pair. The sources of the pair of KGs are DBpedia and Wikidata, so we call them **DW-NB-100K**, **DW-NB-300K**, and **DW-NB-600K**, respectively. We have addressed the bijection and name variety problems in them such that the numbers of seed entity alignments are around 50K, 150K, and 300K, respectively. For each dataset, we

Table 7 Effects of multilingual KGs

Technique	Hits@1			
	Not using attributes		Using attribute	
	EN-DE	EN-FR	EN-DE	EN-FR
MTransE	14.51	8.58	21.78	13.31
IPTransE	8.09	9.45	12.80	14.46
BootEA	24.67	35.20	36.66	51.50
TransEdge	27.53	37.81	40.11	55.21
MuGNN	15.61	19.44	23.22	28.81
Alinet	14.07	18.36	20.82	27.30
KECG	20.90	20.34	31.24	30.28
JAPE	15.86	19.90	24.00	29.44
MultiKE	46.53	41.78	67.56	60.56
AttrE	14.55	12.83	64.74	56.79
GCN-Align	21.18	30.86	31.10	45.38
HGCN	46.78	38.25	67.76	55.91
GMNN	46.53	38.09	67.63	55.28
RDGCN	46.06	39.77	66.72	57.95
CEA	46.72	43.97	67.83	63.83
MRAEA	47.67	43.25	69.13	62.67
NMN	48.08	42.96	69.47	62.72

use 30% of the aligned entities for training. We focus our experiments on four representative techniques, AttrE, MultiKE, NMN, and MRAEA, the top-2 from translation- and GNN-based techniques on the DW-NB dataset.

Theoretical analysis. For simplicity, suppose both KGs have a similar number of entities N. Let M denote the total number of triples (i.e., the number of edges) in the two KGs; then M is N^2 in the worst case but $M << N^2$ in practice as the graph is sparse.

The inference module is typically via NNS or similar operations, which has the time/space cost of $\mathcal{O}(N)$ for each entity; for aligning all the entities, the time cost is $\mathcal{O}(N^2)$ and space cost is $\mathcal{O}(N)$.

The training module includes an embedding module and an alignment module. The time/space cost for the embedding module is $\mathcal{O}(M)$ as it iterates through all the triples in the two KGs as well as a fixed number of negative samples per positive sample (i.e., per triple). The time/space cost of the alignment module depends on the algorithm. Most techniques iterate through the seed entity alignments and optionally a fixed number of negative alignment samples per seed entity alignment, so the cost is $\mathcal{O}(|\mathcal{S}|)$, where $|\mathcal{S}|$ is the number of seed entity alignments. AttrE is a special case as it does not use seeds; its cost is $O(N)$ according to Sect. 5.2. The space cost of alignment is $O(N)$. The space cost for the training is $\mathcal{O}(N)$.

Note. Although translation- and GNN-based techniques have the same asymptotic training cost, their practical GPU memory usage differs greatly due to different training mechanisms. Each training iteration of translation-based techniques typically requires computing the translation function (Eq. 3) or its variant, which only involves several triples. The two KGs and the embeddings are stored in the CPU memory. A machine learning framework such as TensorFlow only loads the triples needed for each training iteration, namely mini-batch, from the CPU memory to the GPU memory. We can control the mini-batch size to be as small as only several triples, so the required GPU memory for translation-based techniques is very small. In comparison, GNN-based techniques usually use message passing to compute the embedding of graph nodes and edges, and the message passing procedure in TensorFlow loads the whole graph into the GPU memory, which is $\mathcal{O}(M)$, a large number. GPU memory is a bottleneck for running machine learning algorithms. This makes translation-based techniques more scalable than GNN-based ones in terms of GPU memory requirement.

Experimental results. Table 8 shows the running time and GPU memory usage of the training module of four representative techniques as data sizes grow.

The running time of all the tested techniques is similar for the same dataset size. Translation-based techniques, AttrE and MultiKE, have constant GPU memory usage, because it is determined by the size of mini-batches. MultiKE uses more GPU memory than AttrE because MultiKE is more complicated and computes more things. As the data size grows from 100K to 300K, the GPU memory usage of GNN-based techniques grows more than translation-based ones because it is $\mathcal{O}(M)$. We are not able to run the two GNN-based techniques on the 600K dataset as the GPU memory on our server (32GB) is not enough. In summary, translation-based techniques are more scalable than GNN-b ased ones in practice. We observe that the growth of runnin g time and GPU memory usage of GNN-based techniques i s sub-linear, which is due to great sparsity of the graphs. Th e inference modules of all the techniques take two to three hours, much less than training.

Table 9 shows the accuracy of the techniques as the dataset size grows. The accuracy of all the techniques degrades a little as the number of entities increases. This is because for the same entity in a KG, there are more entities in the other

Table 8 Running time and GPU memory versus dataset sizes

Technique	Running time (days)			GPU memory usage (GB)		
	100K	300K	600K	100K	300K	600K
AttrE	2.2	5.6	10.9	4.5	4.5	4.5
MultiKE	2.2	6.0	11.5	7.3	7.3	7.3
NMN	2.8	6.1	N/A	11.2	28.6	N/A
MRAEA	2.5	5.7	N/A	11.5	28.6	N/A

Table 9 Accuracy versus dataset sizes

Technique	100K		300K		600K	
	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10
AttrE	75.59	80.30	70.59	74.95	61.22	64.98
MultiKE	75.86	79.86	69.56	72.48	61.42	65.53
NMN	75.18	78.68	70.61	73.28	N/A	N/A
MRAEA	71.70	77.00	69.18	72.33	N/A	N/A

KG similar to it in the case of larger datasets, making it harder to predict the aligned entity correctly.

Alignment on large-scale KGs. An interesting question is whether the algorithms can scale up to large-scale KGs such as the full Wikidata (95 million entities) with Freebase (86 million entities). Interested readers are referred to the longer version of this paper at https://arxiv.org/abs/2103.15059.

8 Conclusions and future directions

We have provided a comprehensive tutorial-type survey on representative EA techniques that use the new approach of representation learning. We have presented a framework for capturing the key characteristics of these techniques, proposed a benchmark DWY-NB to address the limitation of existing benchmark datasets, and conducted extensive experiments using the proposed datasets. The experimental study shows the comparative performance of the techniques and how various factors affect the performance. An insight from the experiments is that making good use of semantic information such as attribute triples improves the accuracy significantly. AttrE and MultiKE consistently perform the best in various settings of our experiments.

Future Directions In terms of the benchmark, more experimental settings may be further explored such as varying the proportion of entities with the same name (i.e., the proportion of the "tricky" feature), and the ratio between relation triples and attribute triples.

In terms of the accuracy of EA techniques, we may improve via pre-training. Pre-training has been very successful in NLP but its use in knowledge bases is limited to using pre-trained word embeddings for initializing entity name features. There is still huge potential of innovative ways of pre-training. For example, pre-trained predicate embeddings may be computed based on the predicate description to capture the semantics and similarity of predicates from different KGs. To train such embeddings, we may use transformer for relation prediction from relation descriptions, i.e., given a relation description, the model is trained to predict the corresponding relation. It may be further expanded into relation prediction between two entities, where the model takes the description of two entities and predicts the relations between the two entities.

Various components and strategies used by EA techniques may be improved following the analysis and discussions based on our framework. First, many translation-based EA techniques uses TransE as the KG structure embedding (cf. Table 2). We may explore replacing this component by improved versions of TransE such as TransD [20] and TransR [30]. Second, many translation-based techniques use the same loss function as TransE (Eq. 3). We may try the limit-based loss function (Eq. 13) which has been reported to have better performance [83]. Third, most of the existing EA techniques use seed alignments in the alignment module (cf. Fig. 2), but seed alignments are expensive and difficult to obtain, so unsupervised EA techniques will be an attractive direction.

In terms of the efficiency and scalability of EA techniques, existing studies have mostly been conducted on small datasets. It is important to develop techniques that can conduct EA on very large KGs, improving on both efficiency and memory required.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: Dbpedia: A nucleus for a web of open data. In: ISWC 2007 (2007)
2. Bhattacharya, I., Getoor, L.: Entity resolution in graphs. Mining Graph Data **13**, 311–344 (2006)
3. Bollacker, K.D., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: SIGMOD 2008 (2008)
4. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NeurIPS 2013 (2013)
5. Cao, Y., Liu, Z., Li, C., Liu, Z., Li, J., Chua, T.S.: Multi-channel graph neural network for entity alignment. In: ACL 2019 (2019)
6. Chen, B., Zhang, J., Tang, X., Chen, H., Li, C.: Jarka: Modeling attribute interactions for cross-lingual knowledge alignment. In: PAKDD 2020 (2020)
7. Chen, M., Tian, Y., Yang, M., Zaniolo, C.: Multilingual knowledge graph embeddings for cross-lingual knowledge alignment. In: IJCAI 2017 (2017)
8. Chen, M., Tian, Y., Chang, K., Skiena, S., Zaniolo, C.: Co-training embeddings of knowledge graphs and entity descriptions for cross-lingual entity alignment. In: IJCAI 2018 (2018)
9. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
10. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL 2019 (2019)
11. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: SIGKDD 2014 (2014)
12. Du, L., Kumar, A., Johnson, M., Ciaramita, M.: Using entity information from a knowledge base to improve relation extraction. In: ALTA 2015 (2015)
13. Färber, M.: The microsoft academic knowledge graph: a linked data source with 8 billion triples of scholarly data. In: ISWC 2019 (2019)
14. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. JASA **64**(328), 1183–1210 (1969)
15. Francois, S., Francois, L.Y., Chuguang, Z.: Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In: IJCAI Workshop 2009 (2009)
16. Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast rule mining in ontological knowledge bases with AMIE+. VLDBJ **24**(6), 707–730 (2015)

17. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: ICML 2017 (2017)
18. Guo, L., Sun, Z., Hu, W.: Learning to exploit long-term relational dependencies in knowledge graphs. In: ICML 2019 (2019)
19. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. Artif. Intell. **194**, 28–61 (2013)
20. Ji, G., He, S., Xu, L., Liu, K., Zhao, J.: Knowledge graph embedding via dynamic mapping matrix. In: ACL 2015 (2015)
21. Ji, S., Pan, S., Cambria, E., Marttinen, P., Yu, P.S.: A survey on knowledge graphs: Representation, acquisition and applications. CoRR arXiv:2002.00388 (2020)
22. Julius, V., Christian, B., Martin, G., Georgi, K.: Discovering and maintaining links on the web of data. In: ISWC 2009 (2009)
23. Kathuria, M., Nagpal, C., Duhan, N.: Journey of web search engines: milestones, challenges & innovations. IJITCS **12**, 47–58 (2016)
24. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR 2017 (2017)
25. Kuhn, H.W.: The hungarian method for the assignment problem. In: 50 Years of Integer Programming 1958–2008 (2010)
26. Kulkarni, S., Singh, A., Ramakrishnan, G., Chakrabarti, S.: Collective annotation of wikipedia entities in web text. In: SIGKDD 2009 (2009)
27. Li, C., Cao, Y., Hou, L., Shi, J., Li, J., Chua, T.: Semi-supervised entity alignment via joint knowledge embedding model and cross-graph model. In: ENLP 2019 (2019)
28. Lin, X., Yang, H., Wu, J., Zhou, C., Wang, B.: Guiding cross-lingual entity alignment via adversarial knowledge embedding. In: ICDM 2019 (2019)
29. Lin, Y., Liu, Z., Luan, H., Sun, M., Rao, S., Liu, S.: Modeling relation paths for representation learning of knowledge bases. In: EMNLP 2015 (2015)
30. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: AAAI 2015 (2015)
31. Liu, Z., Cao, Y., Pan, L., Li, J., Chua, T.: Exploring and evaluating attributes, values, and structures for entity alignment. In: EMNLP 2020 (2020)
32. Mao, X., Wang, W., Xu, H., Lan, M., Wu, Y.: MRAEA: an efficient and robust entity alignment approach for cross-lingual knowledge graph. In: WSDM 2020 (2020)
33. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: ICLR Workshop 2013 (2013)
34. Ngomo, A.C.N., Auer, S.: Limes: a time-efficient approach for large-scale link discovery on the web of data. In: IJCAI 2011 (2011)
35. Nie, H., Han, X., Sun, L., Wong, C.M., Chen, Q., Wu, S., Zhang, W.: Global structure and local semantics-preserved embeddings for entity alignment. In: IJCAI 2020 (2020)
36. Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., Auli, M.: fairseq: A fast, extensible toolkit for sequence modeling. In: NAACL-HLT 2019 (2019)
37. Pei, S., Yu, L., Hoehndorf, R., Zhang, X.: Semi-supervised entity alignment via knowledge graph embedding with awareness of degree difference. In: Web Conference 2019 (2019)
38. Pei, S., Yu, L., Zhang, X.: Improving cross-lingual entity alignment via optimal transport. In: IJCAI 2019 (2019)
39. Qin, K.K., Salim, F.D., Ren, Y., Shao, W., Heimann, M., Koutra, D.: G-crewe: Graph compression with embedding for network alignment. In: CIKM 2020 (2020)
40. Rahimi, A., Cohn, T., Baldwin, T.: Semi-supervised user geolocation via graph convolutional networks. In: ACL 2018 (2018)
41. Raimond, Y., Sutton, C., Sandler, M.B.: Automatic interlinking of music datasets on the semantic web. In: WWW Workshop 2008 (2008)
42. Roth, A.E.: Deferred acceptance algorithms: history, theory, practice, and open questions. Int. J. Game Theory **36**(3–4), 537–569 (2008)
43. Shi, X., Xiao, Y.: Modeling multi-mapping relations for precise cross-lingual entity alignment. In: EMNLP 2019 (2019)
44. Suchanek, F.M., Abiteboul, S., Senellart, P.: Paris: Probabilistic alignment of relations, instances, and schema. In: PVLDB 2011 (2011)
45. Sun, Z., Hu, W., Li, C.: Cross-lingual entity alignment via joint attribute-preserving embedding. In: ISWC 2017 (2017)
46. Sun, Z., Hu, W., Zhang, Q., Qu, Y.: Bootstrapping entity alignment with knowledge graph embedding. In: IJCAI 2018 (2018)
47. Sun, Z., Huang, J., Hu, W., Chen, M., Guo, L., Qu, Y.: Transedge: Translating relation-contextualized embeddings for knowledge graphs. In: ISWC 2019 (2019)
48. Sun, Z., Wang, C., Hu, W., Chen, M., Dai, J., Zhang, W., Qu, Y.: Knowledge graph alignment network with gated multi-hop neighborhood aggregation. In: AAAI 2020 (2020)
49. Sun, Z., Zhang, Q., Hu, W., Wang, C., Chen, M., Akrami, F., Li, C.: A benchmarking study of embedding-based entity alignment for knowledge graphs. In: VLDB 2020 (2020)
50. Tejada, S., Knoblock, C.A., Minton, S.: Learning object identification rules for information integration. Inf. Syst. **26**(8), 607–633 (2001)
51. Trisedya, B.D., Qi, J., Zhang, R.: Entity alignment between knowledge graphs using attribute embeddings. In: AAAI 2019 (2019)
52. Trisedya, B.D.,Weikum, G., Qi, J., Zhang, R.: Neural relation extraction for knowledge base enrichment. In: ACL 2019 (2019)
53. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NIPS 2017 (2017)
54. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR 2018 (2018)
55. Verykios, V.S., Elmagarmid, A.K., Houstis, E.N.: Automating the approximate record-matching process. Inf. Sci. **126**(1–4), 83–98 (2000)
56. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. CACM **57**(10), 78–85 (2014)
57. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: a survey of approaches and applications. TKDE **29**(12), 2724–2743 (2017)
58. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: AAAI 2014 (2014)
59. Wang, Z., Lv, Q., Lan, X., Zhang, Y.: Cross-lingual knowledge graph alignment via graph convolutional networks. In: EMNLP 2018 (2018)
60. Wang, Z., Yang, J., Ye, X.: Knowledge graph alignment with entity-pair embedding. In: EMNLP 2020 (2020)
61. Wu, Q., Shen, C., Wang, P., Dick, A., van den Hengel, A.: Image captioning and visual question answering based on attributes and external knowledge. TPAMI **40**(06), 1367–1381 (2018)
62. Wu, Y., Liu, X., Feng, Y., Wang, Z., Yan, R., Zhao, D.: Relation-aware entity alignment for heterogeneous knowledge graphs. In: IJCAI 2019 (2019)
63. Wu, Y., Liu, X., Feng, Y., Wang, Z., Zhao, D.: Jointly learning entity and relation representations for entity alignment. In: EMNLP 2019 (2019)
64. Wu, Y., Liu, X., Feng, Y., Wang, Z., Zhao, D.: Neighborhood matching network for entity alignment. In: ACL 2020 (2020)
65. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. TNNLS **32**(1), 4–24 (2021)

66. Xiao, H., Huang, M., Zhu, X.: From one point to a manifold: Knowledge graph embedding for precise link prediction. In: IJCAI 2016 (2016)
67. Xiao, H., Huang, M., Zhu, X.: Transg: a generative model for knowledge graph embedding. In: ACL 2016 (2016)
68. Xie, Q., Ma, X., Dai, Z., Hovy, E.H.: An interpretable knowledge transfer model for knowledge base completion. In: ACL 2017 (2017)
69. Xu, K., Wang, L., Yu, M., Feng, Y., Song, Y., Wang, Z., Yu, D.: Cross-lingual knowledge graph alignment via graph matching neural network. In: ACL 2019 (2019)
70. Xu, K., Song, L., Feng, Y., Song, Y., Yu, D.: Coordinated reasoning for cross-lingual knowledge graph alignment. In: AAAI 2020 (2020)
71. Xu, L., Zhou, Q., Gong, K., Liang, X., Tang, J., Lin, L.: End-to-end knowledge-routed relational dialogue system for automatic diagnosis. In: AAAI 2019 (2019)
72. Yang, H., Zou, Y., Shi, P., Lu, W., Lin, J., Sun, X.: Aligning cross-lingual entities with multi-aspect information. In: EMNLP 2019 (2019)
73. Yang, K., Liu, S., Zhao, J., Wang, Y., Xie, B.: COTSAE: co-training of structure and attribute embeddings for entity alignment. In: AAAI 2020 (2020)
74. Yang, S., Zhang, R., Erfani, S.M.: Graphdialog: Integrating graph knowledge into end-to-end task oriented dialogue systems. In: EMNLP 2020 (2020)
75. Yang, S., Zhang, R., Erfani, S.M., Lau J.H.: UniMF: A unified framework to incorporate multimodal knowledge bases into end-to-end task-oriented dialogue systems. In: IJCAI 2021 (2021)
76. Ye, R., Li, X., Fang, Y., Zang, H., Wang, M.: A vectorized relational graph convolutional network for multi-relational network alignment. In: IJCAI 2019 (2019)
77. Yuan, Y., Xiong, Z., Wang, Q.: ACM: adaptive cross-modal graph convolutional neural networks for RGB-D scene recognition. In: AAAI 2019 (2019)
78. Zeng, W., Zhao, X., Tang, J., Lin, X.: Collective entity alignment via adaptive features. In: ICDE 2020 (2020)
79. Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.Y.: Collaborative knowledge base embedding for recommender systems. In: SIGKDD 2016 (2016)
80. Zhang, Q., Sun, Z., Hu, W., Chen, M., Guo, L., Qu, Y.: Multi-view knowledge graph embedding for entity alignment. In: IJCAI 2019 (2019)
81. Zhang, Z., Liu, H., Chen, J., Chen, X., Liu, B., Xiang, Y., Zheng, Y.: An industry evaluation of embedding-based entity alignment. In: COLING 2020 (2020)
82. Zhao, X., Zeng, W., Tang, J., Wang, W., Suchanek, F.M.: An experimental study of state-of-the-art entity alignment approaches. TKDE **2020**, 1–1 (2020)
83. Zhou, X., Zhu, Q., Liu, P., Guo, L.: Learning knowledge embeddings by combining limit-based scoring loss. In: CIKM 2017 (2017)
84. Zhu, H., Xie, R., Liu, Z., Sun, M.: Iterative entity alignment via joint knowledge embeddings. In: IJCAI 2017 (2017)
85. Zhu, Q., Zhou, X., Wu, J., Tan, J., Guo, L.: Neighborhood-aware attentional representation for multilingual knowledge graphs. In: IJCAI 2019 (2019)
86. Zhu, Q., Wei, H., Sisman, B., Zheng, D., Faloutsos, C., Dong, X.L., Han, J.: Collective multi-type entity alignment between knowledge graphs. In: Web Conference 2020 (2020)
87. Zhuang, C., Ma, Q.: Dual graph convolutional networks for graph-based semi-supervised classification. In: Web Conference 2018 (2018)