Duplicate Detection in a Knowledge Base with PIKA

Maxime Prieur, Guillaume Gadek and Bruno Grilheres
Airbus Defence and Space, France

Keywords: Knowledge Base, Entity Embedding, Duplicate Detection, Graph Convolutional Networks.

Abstract: This paper explores the use of Graph Neural Network models producing node embeddings, in order to solve the not fully addressed problem of detecting similar items stored in a knowledge base. Leveraging pre-trained models for textual semantic similarity, our proposed method `PIKA` aggregates heterogeneous (structured and unstructured) characteristics of an entity and its neighborhood to produce an embedding vector that can be used in different tasks such as information retrieval or classification tasks. Our method learns specific weights for each information brought by an entity, enabling us to process it in an inductive fashion.

1 INTRODUCTION

In the last few years, the concept of knowledge bases (KB) has gained a lot of interest. In a KB, the elements sharing relations can be linked among each other, which can then be visualized in the form of a Knowledge Graph, very helpful for tasks like social network analysis, decision making, question answering or product to user recommendation, (Dai et al., 2020; Xiaohan, 2020). Such a database can be constructed and filled either manually (from people in charge of entering day-to-day information) or automatically (from parsing a continuous flow of data). The latter is part of a bigger task called automatic knowledge base population.

In both previous cases, there is a risk of adding an already existing element due to typos in the values of the attributes, several users concurrently updating the knowledge base or an entity being added with up-to-date information without worrying about one already present in the database. These cases are a big concern when administrating the database: they are introducing redundancies, polluting data and impacting the performances on the different tasks. In this paper, we propose a solution to avoid this type of errors by detecting similar entities in a knowledge base before inserting new items.

To the best of our knowledge, this problem has not been fully addressed yet. An appropriate solution should take into account the different types of entities stored in a database (e.g Person, Location, etc...), being able to scale to a large number of elements (billion) with heterogeneous description and taking advantages of both the entity description (attributes) and its neighborhood (relations and attributes of the neighbors).

Several attempts have been made to determinate the similarity between pairs of elements, mostly based on their attributes (Mudgal et al., 2018). However, their way to operate faces a scalability issue when dealing with large numbers of entities since the pairwise measure has to be computed for each candidate.

A few methods have been proposed to produce a vector representation of the elements in a knowledge base for predicting link between entities (Wang et al., 2021; Shi and Weninger, 2018). Still, these approaches are either based on the requirement that all of the entities are associated with a descriptive text or that the entities have no information other than their name. Other approaches adopt the graph neural network framework from (Wang et al., 2019); but as the previous ones, they use a loss based on the probability of a relation between two entities. This is not suited to the problem we are trying to solve and can lower the discriminating power of the information carried by the neighborhood.

In this paper, our main contributions are the following:

- we introduce a pre-processing step to aggregate the attributes of an entity in a unique vector used as an input for embedding models;
- we are the first to propose a fast and reliable GCN model, `PIKA`, capable to produce node embeddings within a heterogeneous graph, in order to retrieve similar or almost duplicate entities;
- we perform an in-depth evaluation and comparison of our proposed model to state-of-the-art

Prieur, M., Gadek, G. and Grilheres, B.
Duplicate Detection in a Knowledge Base with PIKA.
DOI: 10.5220/0010769500003116
In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART 2022) - Volume 3*, pages 46-54
ISBN: 978-989-758-547-0; ISSN: 2184-433X

GCNs methods, retrieving generated clones from the TAC KBP database.

The remaining of this article is structured as follows. Section 2 reviews the related work regarding the embedding of the entities in a knowledge base. Section 3 is a detailed description of our model. Section 4 formalizes the experiment conducted to measure the embeddings quality. Finally, Section 5 shows and describes the obtained results before discussing what could be explored in future works.

2 RELATED WORK

2.1 Entity Matching

The task of entity matching consists in determining the similarity between two records from the same database or different databases (entity alignment); this task can be tackled in a number of ways. A hybrid module that uses the alignment of encoded values of the same attribute between a pair of records has been proposed by (Mudgal et al., 2018). The HighwayNet model is a bi-directional RNN with a decomposable attention mechanism; it summarizes attributes before applying a concatenation and an element-wise absolute difference to produce an output which is then classified (Srivastava et al., 2015).

Two other models, Ditto and Entity Matching on Unstructured Data, proposed respectively by (Li et al., 2020) and (Brunner and Stockinger, 2020), chose to fine-tune pre-trained language models such as Roberta (Liu et al., 2019), giving the concatenation of the two serialized records as inputs. These methods do not integrate the graph aspect of a knowledge base since they are applied to atomic records not linked to each other.

2.2 Relation based Embedding

Node embedding methods use the neighborhood of a node, generating random walks of fixed length around each node to build a context to compute an embedding, as introduced by DeepWalk (Perozzi et al., 2014).This principle of using the context to train encoders relying on the frequency of appearance of the element is frequently exploited (Nielsen, 2017; Ristoski and Paulheim, 2016; Dong et al., 2017).The first disadvantage of this is that embeddings are obtained on a vocabulary (e.g. the nodes in the graph) of fixed size, which rules out any comparison with new nodes. Moreover, relying only on the neighborhood is widely applied to community detection tasks, as described in (Fortunato, 2010), but relevant, cohesive communities can only be detected by considering the network and the attributes (e.g., the node-born semantic information) (Gadek, 2019).

More focused on the existence of a relation between two records, the “translational” approach chooses to model links as translations from one entity to another (Bordes et al., 2013; Dettmers et al., 2018). The embeddings are computed using a margin-base loss function reducing the distance of real triplets. Rotate, from (Sun et al., 2019), deals with the relations as rotation in a vector space to allow symmetry, anti-symmetry, inversion and composition to be taken into account. Once again, the problem of transductivity arises. In addition to this, keeping as a training objective the predictions of the relations is not optimal for the detection of similarities.

2.3 Graph Neural Networks

The recent development of deep-learning architectures, especially convolution layers, has led to an interest to reuse them in graph processing, as illustrated in (Kipf and Welling, 2016). The information around the node can be aggregated and iteratively updated to obtain the representation of a target, in an inductive manner with the Graphsage model (Hamilton et al., 2017). This kind of method is usually limited to a depth of 1 or 2 layers because of an over-smoothing problem (Wu et al., 2019) and to maintain low processing times. Still, GraphSage provides learned reusable weights to handle new nodes.

Since not all neighbors have the same influence on an entity, GAT, a GNN model using linear projections and a self-attention mechanism, has been introduced (Veličković et al., 2017). Arguing that the attention used by GAT is not dynamic and struggles to process the real influence of the nodes, a modification of the sequence of operations of the previous model has been proposed to solve the issue (Brody et al., 2021).

The above-mentioned approaches disregard the heterogeneity of the graph, an aspect that is covered by HinSage, an adaptation of (Hamilton et al., 2017) implemented in StellarGraph[1]. HinSage groups and averages the vectors of the neighborhood according to the type of the relationship or neighbor. The method called HGT computes the attention coefficient of the neighbors, the message passed by them and merges everything into a new vector. All the operations involve class- and relation-specific linear projections. This makes it complex to use in the case of a knowledge graph because of the large number of different edges.

[1]https://stellargraph.readthedocs.io

2.4 Node Embeddings in Knowledge Graphs

Recent work also suggested to combine the translational loss functions from (Bordes et al., 2013) with the fine-tuning of a textual model to encode the description of the entities (Wang et al., 2021; Daza et al., 2020). This approach performs well when provided a rich textual description, which is however not consistently the case with knowledge databases.

2.5 Discussion

Our review of the related work highlights the need for a solution that can:

- Make the most of the additional information brought by the nature of knowledge graphs (i.e. the type of relationships and the class of the nodes);
- Carry out the search quickly enough to be able to propose similar entities in real time or near real-time among thousands of items;
- Operate inductively to allow new nodes to be processed without training over the model and computing one more time the representations of all the nodes;
- Transform the node profile into a vector without loss of content, in the case of sparse or rich attributes.

This paper aims to address these needs by proposing an adaptation of the GCNs models.

3 TASK DESCRIPTION AND MODEL FORMALIZATION

3.1 Problem Definition

A knowledge base is a data structure used to store heterogeneous entities with structured information. This structure can be modeled as an attributed heterogeneous graph, $G = (V, E, \Lambda, M, A)$, where V is the set of nodes (representing entities; the two terms are used interchangeably) constituting the graph, $\varepsilon_{i,j} \in E$ is an oriented link between two nodes, Λ is the association of each entity with a class, M is the association of an edge, $\varepsilon_{i,j}$, to its type and $\alpha(i) = \{\alpha(i)_1, \ldots \alpha(i)_k\} \in A$, is the set of attributes possessed by the node $v(i) \in V$. An attribute is constituted of a name and the embedding of its textual representation, $\alpha(i)_j = (\alpha(i)_{j,name}, \alpha(i)_{j,val})$.

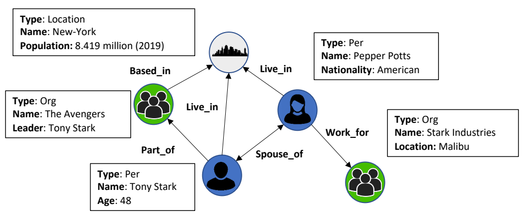

Figure 1: Example of a knowledge graph.

As shown in Figure 1, the difficulties with this type of graph are in the possible variation of attributes between two nodes of the same class (qualified as dirty entity matching) and the number of relations.

3.2 Model Architecture

Our model is composed of two parts: the profile embedding, shown in Figure 2, and the graph model, in Figure 3. Since graph methods require the submission of a vector for each entity, a first step is to summarize the profile information. The second step consists in a graph model adapted to the nature of knowledge graphs. The two parts can be trained separately.

To stabilize and reduce learning time, we use the k-head learning approach introduced by (Vaswani et al., 2017).As described for the GAT model, the k-head learning consists in training several versions of the model parameters in parallel. The resulting representations obtained for each head are then averaged into a single vector.

3.3 Input Computation

For the first stage of embedding, we rely on a pre-trained encoder to transform strings into vectors, such as Universal Sentence Encoder (Cer et al., 2018), but also Mpnet (Song et al., 2020) and DistilRoberta, a lighter model adapted from (Liu et al., 2019). These models were trained to measure the semantic similarity between two sentences. For the last two models we used the instances proposed by the *sentence-transformers*[2] library; they were computed following the process described by (Reimers and Gurevych, 2019).

We selected the Universal Sentence Encoder for a better trade-off between processing time and efficiency. One could argue that attributes in our case are made up of words that moreover are not often found in the literature (First name, Spouse name, etc...) and therefore a character-based encoder like FastText would be more appropriate. As a matter of fact, Fastext is used for those reasons in (Mudgal et al., 2018).

[2]https://www.sbert.net

An exhaustive comparison of which encoder is the best suited for each kind of attribute is left for future work.

Once all the values of the fields constituting an entity are encoded, we need to merge their information. First, attribute-specific coefficients, $w_{\psi(\alpha(i)_{j,name})}$, are used to weight the feature vectors of an attribute, $\alpha(i)$. ψ, being the function mapping the attribute name to his corresponding weight and $\alpha(i)_{j,name}$, the name of the j-th attribute of the node i. Then those weighted vectors are aggregated. For this step we tested several approaches:

- Global Accumulation-Pooling: this operation builds the entity embedding by building-up all his weighted features.
- Max-Pooling: widely used for methods processing images and reused in (Hamilton et al., 2017), this operation returns the maximum value of the inputs, index-wise.
- Extreme-Pooling: since the similarities are measured with vector distances, keeping negative values in the embedding enables a better comparison while keeping the max-pooling operation.

Equation 1 shows how we compute the profile embedding, h_p, of a node i with k attributes, using extreme-pooling. The weights and attribute feature have values in $\mathbb{R}^d$ with d the length of the feature vectors. The $\bullet$ represents the Hadamar product.

$$\begin{aligned} h_p(i) =& extreme(\{w_{\psi(\alpha(i)_{j,name})} \bullet \alpha(i)_{j,val}\}) \\ & for\ j \in \{1,...,k\} \end{aligned} \quad (1)$$

Since the profile does not depend on the relation, once it is computed it only needs to be updated when a change is made to the entity itself. However, the use of specific weights per attribute makes it impossible to infer on unseen field types. In the case where entities have few attributes, the weighting vectors can easily be replaced by matrices.

3.4 Taking Into Account Neighbors Through Graph Neural Networks

We have the intuition that the task would be better solved by taking into consideration not only the node profiles, but also their neighborhoods. To do so we rely on a GNN; the state of the art underlines the relevance of a few architecture types: GAT, GATV2, GraphSage, HinSage, HGT. The points covered by the latter models are summarized in Table 1.

We adapt these models to take as inputs, the vector of the target node and all its neighbors. In the case of a model taking into account the type of the node and the neighborhood, we add an entry corresponding to a vector of indices. Those indices associate the target node to its class and the neighbors to their relation type.

3.5 `PIKA`: Performing Identification of Clones in KnowledgeBase Algorithm

We present here in detail `PIKA`, the model that led to the best performances. It is an adaptation of GAT, adjusted to take advantage of the specifics of knowledge graphs. The followings describe the operations performed to produce the node representation.

$$h(i)_{i,j} = w_{\phi(i,j)} \bullet h_p(j) \quad (2)$$

First, each neighbor of the target node is multiplied in an index-wise fashion with an edge-type specific weight. $h_p(j)$ is the profile representation of the node j, w the weight vector (both $\in \mathbb{R}^d$) and $\phi(i,j)$ is the relation type from the record i to record j. ($\phi(i,j) = \phi(j,i)^{-1}$ is the inverse relation. To enable the consideration of a large number of edge types, we used vectors, since matrices would require the computation of too many parameters. Equations 3, 4, 5 reproduce the operations explained in (Veličković et al., 2017).

In Equation 3, the attention coefficient, $e(i)_{i,j}$, of each edge linked to the node i is computed by multiplying an attention vector, $\beta \in \mathbb{R}^{d*2}$ to the concatenation of the representations of the two entities. These coefficients are then normalized by Softmax in Equation 4.

$$e(i)_{i,j} = LeakyReLU(\beta.[h(i)_{i,j}||h(i)_{i,i}]) \quad (3)$$

$$\beta(i)_{i,j} = \frac{e^{e(i)_{i,j}}}{\sum_{k \in Neigh(i)} e^{e(i)_{i,k}}} \quad (4)$$

The last step is an aggregation followed by a nonlinear transformation to compute the final representation of the node i, $h(i)$. $Neigh(i)$ corresponds to the neighborhood of the node i, plus a relation to itself. As several links can exist between two nodes, the neighbor involved can appear more than once in Equation 5.

$$h(i) = \sigma(\sum_{j \in Neigh(i)} \beta(i)_{i,j} \bullet h(i)_{i,j}) \quad (5)$$

Since the direct neighborhood of a node is sufficient to differentiate between cloned and non-cloned pairs, we limit ourselves to models with only one convolutional layer. This allows us to reduce the memory

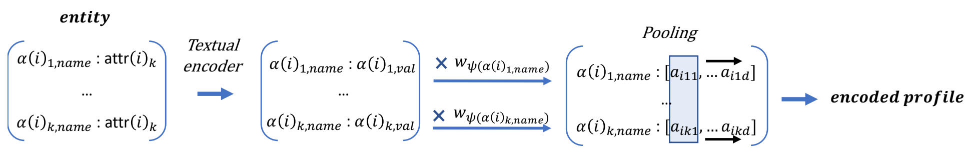
entity
$\alpha(i)_{1,name}$: attr$(i)_k$
...
$\alpha(i)_{k,name}$: attr$(i)_k$
Textual encoder
$\alpha(i)_{1,name} : \alpha(i)_{1,val}$
$\alpha(i)_{k,name} : \alpha(i)_{k,val}$
$\times\ w_{\psi(\alpha(i)_{1,name})}$
$\times\ w_{\psi(\alpha(i)_{k,name})}$
Pooling
$\alpha(i)_{1,name} : [a_{i11}, \dots a_{i1d}]$
$\alpha(i)_{k,name} : [a_{ik1}, \dots a_{ikd}]$
encoded profile

Figure 2: Schema of the profile encoder.

Table 1: Model characteristics.

Model	Directed	Heterogeneous edges	Node type specific	Attention	Iterative inputs
GAT	✗	✗	✗	✓	✓
GATV2	✗	✗	✗	✓	✓
GATV2*	✓	✓	✓	✓	✓
Graphsage	✗	✗	✗	✗	✓
Graphsage*	✗	✗	✓	✗	✓
HinSage	✗	✓	✓	✗	✓
HGT	✓	✓	✓	✓	✓
PIKA-simple	✓	✓	✓	✓	✗
PIKA	✓	✓	✓	✓	✓

used and the time spent on calculating the representation. This makes possible to take into account all the neighbors without the necessity of a sampling strategy as the one used in (Hamilton et al., 2017).

As opposed to the computation of the vector summarizing the profile, if a change occurs at the attributes level, there is a need to recalculate the graph embedding of the neighboring nodes.

3.6 Near-duplicate Finding in Vector Spaces

In order to retrieve the most similar entities based on their embedding we use an approximate nearest neighbor search tool, Annoy[3]. Inspired from kd-trees, it uses random projections and associates a random hyper-plane at each node of the search tree. The drawback of such libraries is the need to update the search index whenever a new node is added.We use the cosine similarity to compare the vectors: it is the most common measure to handle embeddings.

3.7 Training Procedure

3.7.1 Implementation

All the models have been implemented using Pytorch and trained as classifiers in a supervised manner, following the framework explained in (Reimers and Gurevych, 2019). This framework consists in training a binary classifier with pairs of entities as input. This classifier can be used in inference to perform the classification; the last prediction layer can be removed to directly obtain representative vectors, more useful for the entity retrieval task. Our training procedure just changes the output given to the classifier as initially described in the framework. Instead of using the concatenation of the two produced embeddings and their absolute difference, we only keep the difference since it produces better results during our experiments. About the number of attributes, only the 4,000 most present attribute types are taken into account and the 2,000 types of edges (multiplied by two since the direction of the link is considered). The vector shape of the final embeddings is set to 512. Regarding training, we use the binary-cross entropy loss function and the Adam Optimizer ($l_r = 10^{-3}$).

3.7.2 Iterative Training Procedure

The standard training procedure relies on training pairs; the model predicts whether both entities refer to the same or not. We had the intuition that the robustness of the model could be improved here: indeed, the non-analogous pairs are randomly created, and that is why the models can easily differentiate between entities that have nothing in common but could struggle to predict ambiguous entities.

To address this problem, we propose an iterative training procedure in Algorithm 1 which, after each iteration, adds the pairs returned by the model as the most similar. The elements from which the most similar requests are made are the ones in the initial test and validation sets, while the returned instances are the ones that are not already in one of the sets. Note that this procedure is used for the profile encoder step; using it on the neighborhood aggregation step would require first to generate a train/test split with a specific consideration (to avoid having a same neighborhood being in both the train and test sets).

[3]https://github.com/spotify/annoy

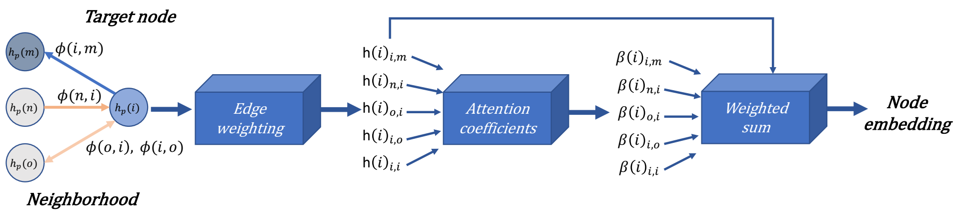

Figure 3: Schema of the `PIKA` model.

Algorithm 1: Iterative training procedure.

```
Input : Set baseTrain, BaseVal
Result: the trained model
train ← baseTrain ;
val ← baseVal ;
model ← new Model ;
while nbIteration ¡ maxIteration do
    model ← train(model, train, val);
    embeddings ← get_embeddings(model);
    most_similar_pairs ←
      get_similarities(embeddings, baseTrain,
      basicVal);
    train, val ← get_new_sets(model, train,
      val);
end
```

4 EXPERIMENT SETTINGS

4.1 The Datasets

In order to develop, test and evaluate the models we used the TAC KBP Reference Knowledge Base developed by the Linguistic Data Consortium (LDC). The knowledge base is built from English Wikipedia articles. More than 800,000 entities are divided into four classes: Person (**PER**), Geo-political entity (**GPE**), Organization (**ORG**) and Unknown (**UKN**). TAC can be modeled as a knowledge graph by using as relations the references to other entities in descriptive texts.

Generating Clones. As TAC is considered to be free of duplicate entities, it was necessary to generate clones from instances of the base. To do so, we used a custom script which generates from a model entity drawn randomly (that has not already been used) a clone with alterations to relationships and attributes. These alterations reproduce at best what can be found in our use cases and have arbitrary probabilities of occurrence. Concerning the attributes, field values can be swapped, the values can be randomly truncated, changes/ insertions of characters may occur to simulate typing errors, attributes can be deleted. About

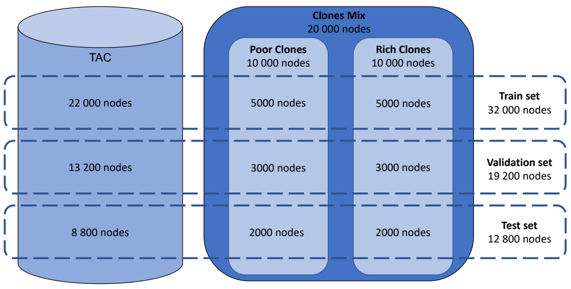

Figure 4: Distribution of the dataset.

the edges, some can be deleted, added with a random pair of neighbors and predicates or existing ones can have the type of the relationship replaced. To prevent bias in the evaluation, an entity can only be used to produce one clone and a clone cannot be used as a model.

To test thoroughly the methods, two types of clones were generated. The first one with few deleted attributes (deletion probability set to 0.2), qualified as **rich** clones and corresponding to an exact clone search. The second type, with a lot of deletion (deletion probability set to 0.5) for the case of a query search and will be named as **poor** clones. The other changes have the same probability of occurrence for both types.

Distribution of the Datasets. As shown in Figure 4, the datasets are made of pairs of nodes "$(entity_A, entity_B)$", associated with a "cloned" or "non-cloned" label. There are respectively 16,000, 9,600 and 6,400 pairs in the training, validation and test set. The datasets are made of 50% cloned pairs. 20% of false clone pairs have one of the entities with a duplicate in the database to prevent from bias in the training procedure. This distribution sets the number of duplicates at 20,000, i.e. 2.5% of the base: it is a trade-off between training set size and a polluted base.

4.2 Evaluation Procedure

The methods are evaluated according to two settings in order to model the ability of a method to operate by proposing similar entities or directly making a classification:

- Search for similar entities: as a problem of information retrieval, for all similar pairs in the test set, we submit the first entity of the tuple as a query. The position of the analogue entity in the list ordered by similarities with the embedding of the query node is then measured.
- Clone detection: we take one of the entities of each pair of the test set. The system then returns the entity it considers the most similar and finally applies a binary classification.

4.3 Metrics

In the context of the search for similar entities, we used the $Hit@k$ score (with $k \in (1, 10, 100)$) and the MRR (Mean Reciprocal Rank). The $Hit@k$ score measures the average capacity of a system to return the target in the first k elements, while the MRR is used to evaluate the performance of the system is general.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{6}$$

$$Hit@k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \mathbf{1}_{target_i \in top_k} \tag{7}$$

The clone detection task is, on the other hand, seen as a classification problem. Given a pair of entities as inputs, the classifier predicts whether both represent the same real entity ("clone") or not. The evaluation metrics are Accuracy, Precision, Recall and F1-score.

Note that this problem is a classification task on top of the IR chain: the classifier examines the submitted entity and its closest match. As a result the recall of the classifier *is* the $Hit@1$ of the IR task.

5 RESULTS AND DISCUSSION

We compare our proposed method PIKA with versions of GraphSage, HGT, HinSage, GAT and GATV2 which we implemented under Pytorch and trained under the (Reimers and Gurevych, 2019) framework. In addition to being tested with their standard implementation, some modifications are added to take into account the specificities of Knowledge Databases. We also measure the performance of PIKA without the iterative training procedure under the label of PIKA-simple, in order to evaluate the added value of this procedure.

The introduction of iterative training really boosts the quality of the model: PIKA clearly outperforms PIKA-simple, its non-iterative implementation. This gain is even greater when switching from basic models to models modified to consider the characteristics of a knowledge graph (noted with a *). Indeed, we can clearly see a better MRR for *-versions, with an exception for GATV2-* on rich clones. The improvement is due, firstly, to the fact that the orientation of the edges is included, which is an interesting parameter for places for example (GPE type). Nodes like these, that are often linked to a myriad of incoming neighbors, would be better represented by their outgoing edges. Secondly, our methods differentiate the weights according to the class of the node in the self-attention mechanism. For some types it is preferable to keep more of the node information. Lastly, distinguishing the nature of connections gives a better adaptability to the model for different topologies.

From the previous results we can also see that models not adapted to the knowledge graph structure (node-type information, relation-type information) struggle to rely on the information given by the neighborhood as their MRR drops between 0.1 and 0.15 points when comparing the results of the same model on the mixed and poor set. Considering the sparsity of the knowledge base (a median of three edges per node and 14% of them do not possess any link), we believe that methods strongly relying on context will perform even better on denser graphs.

As shown in the classification part of the Table 2, the proportion of false positives is still too high (shown as low precision scores) to enable an *automatic* similarity detection system. This can be explained by the fact that no iterative training was done on the classifier with the embedding obtained by the graph methods. Indeed, an iterative training on the graph methods would not have been useful: many nodes have a common neighborhood without being clones. Ambiguity would have hindered iterative learning.

About the time constraint of embedding methods using graphs, we measure an average of 0.16 s required to compute the embedding of an entity and retrieve the most similar item from our knowledge base; this time is computed in a standard office workstation, against the whole TAC KBP database. This short duration makes it convenient to integrate PIKA into Web-scale systems.

6 CONCLUSION

We presented a two-part model that is able to summarize the features and the neighborhood of a node into a vector usable to compare the similarities and to retrieve close entities in terms of characteristics and

Table 2: Measures on the sets of poor clones, then rich clones for the information retrieval task, and on rich clones for the classification task. Models followed by "*" are variants adapted to knowledge graphs.

	Poor clones			Rich clones			Rich clones - classification			
Model	MRR	Hit@1	Hit@100	MRR	Hit@1	Hit@100	Acc.	Precision	Recall	F1
GAT	0.257	0.206	0.534	0.532	0.472	0.764	0.320	0.258	0.957	0.406
GATV2	0.636	0.582	0.850	0.810	0.774	0.936	0.542	0.491	0.924	0.641
GATV2*	0.720	0.678	0.886	0.752	0.711	0.903	0.565	0.513	0.978	0.673
Graphsage	0.449	0.388	0.740	0.783	0.752	0.913	0.431	0.423	**0.993**	0.593
Graphsage*	0.484	0.432	0.764	0.820	0.793	0.928	0.431	0.423	**0.993**	0.593
HinSage	0.110	0.091	0.235	0.521	0.489	0.699	0.764	0.472	0.705	0.566
HGT	0.141	0.119	0.281	0.450	0.426	0.600	0.564	0.297	0.890	0.446
PIKA-simple	0.817	0.783	0.928	0.918	0.899	0.971	0.653	0.617	0.960	0.751
PIKA	**0.856**	**0.828**	**0.941**	**0.959**	**0.949**	**0.988**	**0.786**	**0.669**	0.952	**0.786**

relations. Since the model weights are not specific to nodes but to attributes and relationships, it is applicable to other graphs with the same types of attributes and edges.

For a more specific application, our method could easily be adapted to assign higher weights to attributes that should be focused on. By submitting pairs of clones with specific modifications during training, the user would be able to detect similarities only on targeted attributes.

We obtained a more robust model to ambiguous entity pairs, by applying an iterative training. Nevertheless, embedding a complex entity prevents an efficient atomic comparison of its attributes (as is performed in entity alignment methods) since the information is smoothed into a single vector. Still, the results showed that search entities, if not returned in first position, are almost always in the top 100 results. This considered, we can easily imagine the use of our model as a pre-filtering step on a large database before performing an entity-by-entity comparison or using a more accurate but slower method on fewer candidates.

Finally, the numerical values are not processed in an optimal way by the encoder since they appear in the database as strings. Text encoders such as USE or Mpnet do not specifically handle numerical values: we believe that a dedicated processing for such attributes is required in order to reach an acceptable quality for the clone classification task.

REFERENCES

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

Brody, S., Alon, U., and Yahav, E. (2021). How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491.*

Brunner, U. and Stockinger, K. (2020). Entity matching with transformer architectures-a step forward in data integration. In *International Conference on Extending Database Technology, Copenhagen, 30 March-2 April 2020*. OpenProceedings.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Céspedes, M., Yuan, S., Tar, C., et al. (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175.*

Dai, Y., Wang, S., Xiong, N. N., and Guo, W. (2020). A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5).

Daza, D., Cochez, M., and Groth, P. (2020). Inductive entity representations from text via link prediction. *CoRR*, abs/2010.03496.

Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. (2018). Convolutional 2d knowledge graph embeddings.

Dong, Y., Chawla, N. V., and Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144.

Fortunato, S. (2010). Community detection in graphs. *Physics reports*, 486(3-5):75–174.

Gadek, G. (2019). From community detection to topical, interactive group detection in online social networks. In *IEEE/WIC/ACM International Conference on Web Intelligence-Companion Volume*, pages 176–183.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907.*

Li, Y., Li, J., Suhara, Y., Doan, A., and Tan, W.-C. (2020). Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584.*

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692.*

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. (2018). Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34.

Nielsen, F. Å. (2017). Wembedder: Wikidata entity embedding web service. *arXiv preprint arXiv:1710.04099*.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Ristoski, P. and Paulheim, H. (2016). Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer.

Shi, B. and Weninger, T. (2018). Open-world knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2020). Mpnet: Masked and permuted pre-training for language understanding. *arXiv preprint arXiv:2004.09297*.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, P., Han, J., Li, C., and Pan, R. (2019). Logic attention based neighborhood aggregation for inductive knowledge graph embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7152–7159.

Wang, X., Gao, T., Zhu, Z., Zhang, Z., Liu, Z., Li, J., and Tang, J. (2021). Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR.

Xiaohan, Z. (2020). A survey on application of knowledge graph. *Journal of Physics: Conference Series*, 1487:012016.