CS 245 Project Report

Indie Hackers

December 2021

1 Introduction

Knowledge graphs are used as a means of representing real world information via entities, relationships, and attributes. Information is typically stored as triples, either a relation triple or an attribute triple. Relation triples take the form (entity$_1$, relation, entity$_2$) and attribute triples follow the form (entity, attribute, value). Graphs are denoted as (E, R, A, V, T_R, T_A) where E is the set of all entities, R is the set of all relations, A is the set of all attributes, V is the set of all values, $T_R \in E \times R \times E$ is the set of all relation triples, and $T_A \in E \times A \times V$ is the set of all attribute triples.

Given a set of recognized entities and extracted relations from a document, we can trivially construct a knowledge graph. These knowledge graphs can then be used to map entities and relations into their own vector/embedding space, and clustering processes are run to find representative entities and relations.

Lastly, we'll formally introduce the task of cross-lingual entity alignment which involves aligning entities across multiple monolingual knowledge graphs. The task is to match entities in one graph to their corresponding entities in another graph of a different language base. These entity alignments, or anchor links, can be extremely useful in several downstream applications like language translation or filling knowledge graphs.

1.1 Problem statement

The overall task aims to model and predict the spread, detection and monitoring of risk factors related to Covid-19 and also evaluate the effectiveness across dimensions (time and space). More specifically, we aim to discover anchor links in the cross-lingual entity alignment setting to generate rich embeddings which can be used in downstream tasks to monitor the pandemic and identify risk factors.

Our model will first map entities and relations into an embedding space and then apply a clustering process in the embedding space to identify anchor links between knowledge graph inputs. Given different knowledge graphs as input to our model and a subset of seed anchor links as training data in the supervised setting, we aim to output the aligned entity pairs between these graphs.

2 Related Work

Literature in the knowledge graph alignment space can be categorized into first and second generation methods.

2.1 First generation methods

First generation methods typically exploit distance based embedding techniques. These techniques generally stem from TransE [1], which embeds entities in a vector space and models relations as the vector distance between related entities.

1. **BootEA** [6] uses TransE to generate entity and relation embeddings, then uses a set of seed alignments to reconcile embeddings in a joint semantic space. Further alignment editing helps to reduce errors.

2. **JAPE** [5] generates structural embeddings using a TransE model and attribute embeddings using a skip-gram model, then combines the embedding information to compute similarity scores between the entities.

2.2 Second generation methods

Second generation methods differ from earlier approaches in the embedding problem space in that they favor graph neural networks rather than distance based methods.

1. **GCN-align** [9] uses a graph convolutional network (GCN) to produce structural and attribute embeddings, which are then used along with seed alignments to discover new anchor links.

2. **RDGCN** [10] uses multi-layer GCNs with highway gates to embed networks. Alignment results are obtained by directly comparing the embeddings.

All methods notably require pre-aligned entity pairs as seed alignments or training data. The EMGCN approach proposed in [2] is entirely unsupervised, and thus does not require any entity pairs to be pre-aligned.

3 Approach and implementation

3.1 Approach

Given knowledge graphs from the datasets as input, our methodology first needs to find a succinct embedding that preserves information present in the knowledge graphs regarding entities and their relations. Next, we must investigate how to discover representative entities and relations by employing clustering algorithms in the embedding and attribute space. In the implementation section, we cover the architecture of a baseline model and several of the novel aspects from our work.

3.2 Implementation

We base our implementation on EMGCN and modify different components such as word embedding, embedding model, loss function, and clustering. All of our code is public (see Section 7), and we center our implementations around our overall approach.

3.3 EMGCN

EMGCN has a multi step approach to taking in textual knowledge graphs and finding alignments between them. First, the knowledge graphs must be passed through some word embedder so that textual information is encoded as vector information. Next, the knowledge graphs themselves can be embedded, by applying some embedding model on the new vector representation of the knowledge graphs. This embedding model is trained with a loss function so that embeddings can find relational data. Given two knowledge graph embeddings, a (relational) similarity matrix can be created, which is used in the final alignment procedure. EMGCN also finds these similarity matrices for attributes and values without the use of an embedding model. The knowledge graphs are translated, and then their attributes (or values) are compared via string similarity and set similarity measures to produce the attribute and value similarity matrices. Finally, these matrices are weighted and summed together to produce a single similarity matrix which is then used to create alignments [4].

In addition, this work also provided additional ablation tests on the noise of attribute words compared with several other variations. These variations include attribute information (word embedding of attribute names), value information (mean of word embedding of attribute names), stability refinement, and altering the attribute dictionary. The authors also showed that attribute information should be used carefully because of noise in dataset [4]. This provides valuable insight for clustering and improving the attribute space.

3.4 Activation Function

We also experimented by testing different activation functions in the GCN-based embedding network. The authors [4] propose the following message-passing scheme between GCN layers:

$$H^{(l)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{M}\hat{D}^{-\frac{1}{2}}H^{(l-1)}W^{(l)})$$

where σ is the tanh activation function, $H^{(l-1)}$ is the embedding at the previous layer, $W^{(l)}$ is the weights at the current layer, $\hat{M}$ is an adjacency matrix, and $\hat{D}$ is a diagonal matrix constructed from $\hat{M}$.

We investigate replacing the tanh activation function with ReLU activations and leaky ReLU with a slope of 0.1 for negative values.

3.5 Word embedding

The first step is to take the text data and encode it as a numerical representation. Word embeddings are employed by the GCN to embed the attributes into a vector representation, and this step is also important for representing our attribute names and attribute alignment process.

We explore several methods such as GloVe, fasttext, Gensim Continuous Skipgram (GCS) and BERT. All of these (except BERT) have pretrained word representations, so we can easily change embeddings without extra implementation costs. We expect that using better representations that emphasize both contextual and semantic relations can help us to generate better attribute dictionaries, attribute alignments, and embedding representations in our graph network.

3.6 Embedding model

After each word is embedded, we can then embed the knowledge graph itself. The standard mechanism for this is the GCN architecture.

$$H^{(l)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{M}\hat{D}^{-\frac{1}{2}}H^{(l-1)}W^{(l)})$$

So, the next layer depends on some activation function f, the diagonal matrix $\hat{D}$ (created by summing across columns of $\hat{M}$), the adjacency matrix with self loops $\hat{D}$, the previous layer's information H^{l-1}, and weights for that layer, W^{l}. $H^0 = X$ is the feature matrix given from the input knowledge graph. Activation functions are tanh. The GCN update rule exists in this form due to message passing, since it allows neighbors to propagate information to each other. So deeper layers will have richer information at the cost of requiring more weight parameters and more noise. We also test with adding an additional FC layer after the GCN and by using a graph attention network.

By adding the FC layer, we allow each node to convey info to every other node (instead of only to their neighbors) so we are capable of generating better results, but we may have more bias to the training set or some overfitting in comparison to the graph convolution layers.

In addition, we also try another form of representing our entities through the Graph Attention Network proposed in [7], which leverages masked self-attentional layers to address some of GCNs shortcomings (i.e. smoothing, and not injective). By stacking layers in which nodes are able to attend over their neighborhoods' features, we enable (implicitly) specifying different weights to different nodes in a neighborhood, without requiring any kind of costly matrix operation (such as inversion). We will also mention future directions for better/alternative entity representation.

3.7 Loss

We explore different variations for the loss as well, namely an unsupervised, a supervised and a hybrid loss. The unsupervised loss function from [2] is shown below for clarity.

$$L_{unsupervised} = \sum_{l=1}^{k} ||\hat{D}^{-\frac{1}{2}}\hat{M}\hat{D}^{-\frac{1}{2}} - H^{(l)}H^{(l)T}||_F$$

The unsupervised loss minimizes differences in layer outputs to the normalised Laplacian matrix. By doing this, we train on the structure of the knowledge graph and format the data in a relational manner. Note that the summation takes all intermediate layer outputs, instead of just the last layer. In this way, we can incorporate a multi-order approach.

In addition, because the overall COVID prediction task can be geared towards supervised datasets, as opposed to semi-supervised or unsupervised, we also integrated a supervised loss function. We adapted the supervised loss function from [9] to train the Multi-Order GCN. This adapted loss function is shown below:

$$L_{supervised} = \sum_{(e,v)\in S} \sum_{(e',v')\in S} [f(h(e), h(s)) + \gamma - f(h(e'), h(v'))]$$

Here, $\gamma = 3$ is a hyperparameter, h represents our embedding network for the source and target graphs, (e, v) represents a positive anchor link, and (e', v') represents a negative anchor link produced by randomly corrupting an entity in a true anchor link.

Additionally, we implemented a hybrid loss function that allows us to vary the weights of the supervised and unsupervised loss terms. This loss is shown below:

$$L = \alpha L_{supervised} + (1 - \alpha)L_{unsupervised}$$

where α is a hyperparameter ranging between 0 and 1. Our results for varying α are shown in the experiments section. Note that $\alpha = 0$ simplifies to a purely unsupervised approach and $\alpha = 1$ simplifies to a purely supervised approach. The hybrid loss combines the above two loss terms using a weighted sum and allows us an opportunity to measure the trade-off and usefulness of the supervised-loss and unsupervised-loss.

3.8 Similarity matrix

Once we have trained the embedding network, we can then produce a relational similarity matrix. First, we create $S^l = H_s^l {H_t^l}^T$, a similarity matrix for each output layer, where H_s and H_t denote the source and target knowledge graph hidden layers, respectively. Note here that a similarity matrix is one where S_{ij} represents the similarity score between entity $e_i \in E_{source}$ and $e_j \in E_{target}$. Therefore, we take the elements with high similarity and align them. Then, S^{rel} can be created as a sum of the S^l, i.e. $S^{rel} = \sum_l S^l$. Looking back at the unsupervised loss function, it is reasonable to see why it has this form, as for equal source and target knowledge graphs and a sufficiently trained network,

$S^l \approx \hat{D}^{-\frac{1}{2}}\hat{M}\hat{D}^{-\frac{1}{2}}$, i.e. we would align perfectly. This can be extended to show that permutations are also aligned perfectly.

Since this similarity matrix is created from layer outputs, there is some noise, especially in later layers if we train too much and overfit. To reduce this noise, we employ a refinement procedure: First, find stable nodes, which are defined as any set of entities that are aligned the same way among all S^l. In other words, entity $e_i \in E_{source}$ and $e_j \in E_{target}$ are stable iff S^l_{ij} is the maximal (compared to other entities) for all l. We define all other entities as unstable. Using these stable nodes, the diagonal matrix D is weighted further and thus H^l, S^l are reproduced. This is one refinement iteration, where the number of refinements is a hyperparameter.

Because this similarity matrix is produced only from the word embeddings of relational data in the source / target knowledge graphs, we refer to it as the relational similarity matrix S^{rel}. The attribute and value similarity matrices are constructed without the use of an embedding model as described below.

For attribute similarity, we first translate all attributes and then for each entity we find string-similar attributes in the other knowledge graph and compile them into $A_{dict}(e)$, a set of all other entities with similar attributes. We use a string similarity measure like Sorensen-Dice which compares the bigrams of the words, and take all entities with scores above a certain threshold. Then, similarity matrix can be constructed by $S^{att}_{ij} = J(A_{dict}(e_i), A_{dict}(e_j))$, where $e_i \in E_{source}$ and $e_j \in E_{target}$ and J is a set similarity measure such as Jaccard or Cosine. The S^{val} matrix is created with the values used instead of attributes. Finally, for some hyperparameters $\theta^{att}, \theta^{rel}, \theta^{val}$, we have our similarity matrix S as defined below:

$$S = \theta^{att}S^{att} + \theta^{rel}S^{rel} + \theta^{val}S^{val}$$

3.9 Clustering

We improve the representation/learning process by applying word representations to attributes (dictionary generation) and incorporate clustering and similarity metrics to find better attributes to incorporate. First, we tested the suggested Jaccard similarity measure and also incorporated Cosine similarity over sets.

We also implemented additional clustering measures, such as pruning the overall common attributes used for the alignment matrix. We also incorporate words without embedding representations by default. Because of EMGCN's ablation tests finding the noise impact of attributes, we wanted to cluster the most similar attributes to each source and entity and treat these as "informative" attributes. On the other hand, the bottom γ percent would be pruned from incorporation, where $\gamma \in [10, 20]$. The overall similarity score is calculated as:

$$(\alpha) * \sum_{c \in ca} m_{se} - emb_c + (1 - \alpha) * \sum_{c \in ca} m_{te} - emb_c$$

where α is a hyperparameter that specifies the emphasis on similarity towards the mean source embedding attributes or on similarity towards the mean target embedding attributes. m_{se} stands for mean source attributes embedding, while m_{te} is for mean target attributes embeddings. Lastly, emb_c represents the embedding for our current attribute in the common attributes (ca).

We also added a new similarity method, instead of Jaccard set similarity, which embeds each word and then clusters source nodes to their nearest target nodes. From the clustering, we find a normalised distance measure which represents the similarity. Due to time issues stemming from the $O(n^2)$ time complexity of this clustering method, we were unable to test this approach with our limited compute power.

In general, clustering is applied broadly within the our framework, whereas the similarity matrix is clustering for the first nearest neighbors.

4 Evaluation

4.1 Data

We evaluated our model's performance on the DBP15K dataset. Because this dataset was curated for the task of entity alignment, it will provide meaningful metrics that will help us to gauge the performance of our approach.

The DBP15K dataset consists of four language-specific knowledge graphs (English, Chinese, French, and Japanese) that are extracted from DBpedia. Each knowledge graph contains between 65k and 106k entities. In the alignment task, three sets of 15k alignment labels are constructed to align entities between English and each of the other three languages. These three resulting datasets, namely ZH-EN, FR-EN, and JA-EN, can be used along with the 15k ground truth alignment pairs to train models on the entity alignment task.

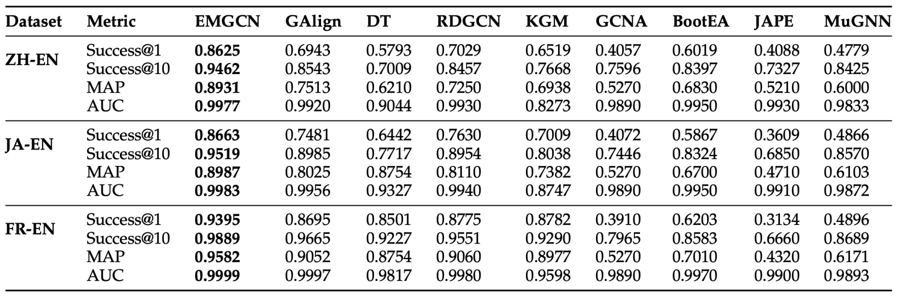
4.2 Experimental Setup

All experiments were conducted on the Chinese-English dataset from DBP15K. We expect similar results on the French-English and Japanese English datasets as well, although compute and memory limitations prevented us from obtaining qualitative evaluations.

For evaluation metrics, we measure the top-k precision and mean average precision (MAP). Top-k precision evaluates the proportion of correctly aligned entities ranked in top k candidates and we evaluate for $k = 1, 10, 50, 100$. MAP assesses how highly ranked the true anchored links are in the list of alignment candidates.

4.2.1 Baseline Comparison

We include baseline results [2] for related work on three datasets. Because the mentioned EMGCN approach outperforms all other baselines methods, we will avoid comparing against these other baseline methods for clarity. Compared to the other baseline methods, EMGCN is the second fastest in terms of runtime. Although GCNA [9] has a faster runtime, it sacrifices performance for speed, as shown in Figure 1.



Dataset	Metric	EMGCN	GAlign	DT	RDGCN	KGM	GCNA	BootEA	JAPE	MuGNN
ZH-EN	Success@1	**0.8625**	0.6943	0.5793	0.7029	0.6519	0.4057	0.6019	0.4088	0.4779
	Success@10	**0.9462**	0.8543	0.7009	0.8457	0.7668	0.7596	0.8397	0.7327	0.8425
	MAP	**0.8931**	0.7513	0.6210	0.7250	0.6938	0.5270	0.6830	0.5210	0.6000
	AUC	**0.9977**	0.9920	0.9044	0.9930	0.8273	0.9890	0.9950	0.9930	0.9833
JA-EN	Success@1	**0.8663**	0.7481	0.6442	0.7630	0.7009	0.4072	0.5867	0.3609	0.4866
	Success@10	**0.9519**	0.8985	0.7717	0.8954	0.8038	0.7446	0.8324	0.6850	0.8570
	MAP	**0.8987**	0.8025	0.8754	0.8110	0.7382	0.5270	0.6700	0.4710	0.6103
	AUC	**0.9983**	0.9956	0.9327	0.9940	0.8747	0.9890	0.9950	0.9910	0.9872
FR-EN	Success@1	**0.9395**	0.8695	0.8501	0.8775	0.8782	0.3910	0.6203	0.3134	0.4896
	Success@10	**0.9889**	0.9665	0.9227	0.9551	0.9290	0.7965	0.8583	0.6660	0.8689
	MAP	**0.9582**	0.9052	0.8754	0.9060	0.8977	0.5270	0.7010	0.4320	0.6171
	AUC	**0.9999**	0.9997	0.9817	0.9980	0.9598	0.9890	0.9970	0.9900	0.9893

Figure 1: Baseline comparisons from [2]

4.2.2 Activation Functions

We investigated modifying the activation function, by evaluating performance with ReLU activations and with leaky ReLU ($alpha = 0.1$) activations. The results are shown below in Figure 2.

		tanh	ReLU	Leaky ReLU
Chinese to English	Hits@1	**0.8640**	0.7947	0.8452
	Hits@10	**0.9512**	0.9177	0.9499
	Hits@50	0.9785	0.9692	**0.9825**
	Hits@100	0.9847	0.9853	**0.9900**
	MAP	**0.8962**	0.8388	0.8832
English to Chinese	Hits@1	**0.8102**	0.7511	0.7936
	Hits@10	**0.9343**	0.8990	0.9297
	Hits@50	0.9735	0.9651	**0.9773**
	Hits@100	0.9840	0.9834	**0.9896**
	MAP	**0.8546**	0.8051	0.8422

Figure 2: Activation Function Results

Replacing the tanh activation function from the baseline with ReLU negatively affects the performance, as does replacing with a leaky ReLU, although to a lesser extent. This performance difference is caused by the loss of information on the negative side of the ReLU function. While a leaky ReLU slightly improves performance over ReLU, the tanh baseline remains the best choice of an activation function due to its bijective nature.

4.2.3 Embedding Functions

We investigated how altering the word embeddings used might affect model performance. These results are shown in Figure 3.

		Glove	FastText	GCS
Chinese to English	Hits@1	0.8640	**0.8754**	0.8645
	Hits@10	0.9512	0.9516	**0.9634**
	Hits@50	0.9785	**0.9798**	0.9756
	Hits@100	0.9847	**0.9910**	0.9849
	MAP	0.8962	**0.8978**	0.8364
English to Chinese	Hits@1	0.8102	0.8045	**0.8161**
	Hits@10	0.9343	0.9343	0.9343
	Hits@50	0.9735	0.9753	**0.9841**
	Hits@100	0.9840	0.9847	**0.9901**
	MAP	0.8546	0.8538	**0.8629**

Figure 3: Embedding Function Results

After experimenting will all the variations of pretrained word embeddings, we found better results with these more contextualized representations (i.e. FastText, GCS, and BERT). Although we were only able to implement and run the BERT version for an experiment, the training time and memory usage made it an infeasible model to test (due to time constraints). This is expected because better representations allows us to improve the attribute alignment process with more representative and similar words. As word embeddings continue to improve, we can expect similar performance improvements to follow.

4.2.4 Loss functions

In this section we investigate how incorporating supervision affects performance. Results are shown in Figure 4.

		$\alpha = 0$	$\alpha = 0.2$	$\alpha = 0.5$	$\alpha = 0.8$	$\alpha = 1$
Chinese to English	Hits@1	0.8640	**0.8796**	0.8761	0.8592	0.8384
	Hits@10	0.9512	**0.9600**	0.9581	0.9491	0.9387
	Hits@50	0.9785	**0.9851**	**0.9851**	0.9804	0.9727
	Hits@100	0.9847	0.9891	**0.9899**	0.9882	0.9814
	MAP	0.8962	**0.9091**	0.9060	0.8919	0.8746
English to Chinese	Hits@1	0.8102	**0.8338**	0.8287	0.8130	0.7914
	Hits@10	0.9343	**0.9470**	0.9457	0.9362	0.9240
	Hits@50	0.9735	**0.9825**	0.9823	0.9762	0.9693
	Hits@100	0.9840	**0.9904**	0.9903	0.9879	0.9818
	MAP	0.8546	**0.8754**	0.8708	0.8569	0.8391

Figure 4: Loss Function Results

Regarding the effects of different hybrid loss values, we observe that a mix of supervised and unsupervised loss yields higher performance. On average, the training time for our supervised loss required 1-2 seconds of compute while unsupervised loss averaged 50-60 seconds. We can also observe the tradeoff in the $\alpha = 0$ yields higher performance than $\alpha = 1$ due to more robust embeddings in favor of training speed. Thus, when we have higher importance for unsupervised, we may yield additional representations from our supervised technique (i.e. $\alpha = 0.2$).

4.2.5 FC layer and Cosine Similarity

In this section we explore using a fully connected layer and cosine similarity. Results are shown in Figure 5.

		Baseline	FC	FC + Cosine
Chinese to English	Hits@1	0.8640	0.8667	**0.8730**
	Hits@10	0.9512	0.9558	**0.9586**
	Hits@50	0.9785	0.9825	**0.9836**
	Hits@100	0.9847	0.9870	**0.9881**
	MAP	0.8962	0.8997	**0.9045**
English to Chinese	Hits@1	0.8102	0.8125	**0.8126**
	Hits@10	0.9343	0.9390	**0.9389**
	Hits@50	0.9735	0.9782	**0.9789**
	Hits@100	0.9840	**0.9869**	0.9868
	MAP	0.8546	0.8578	**0.8586**

Figure 5: FC and Cosine Similarity Results

The results show that the FC + Cosine does outperform the baseline, but not significantly. The idea of the FC layer is that all nodes in the graph can communicate with all other nodes, which can train to produce better results than a regular graph convolution layer. However, the downside is because there are no constraints it may overfit. Here due to the refinement procedure we can avoid overfitting, so in theory this extra FC layer can produce better results. However, the data suggests it is not efficient to add an extra layer for $< 1\%$ gain.

4.2.6 Attention-based methods

In this section we explore how using an attention model to embed information instead of a GCN would affect performance. Results are shown in Figure 6.

		Baseline GCN	Attention
Chinese to English	Hits@1	**0.8640**	0.8530
	Hits@10	**0.9512**	0.9508
	Hits@50	0.9785	**0.9786**
	Hits@100	0.9847	**0.9849**
	MAP	**0.8962**	0.8887
English to Chinese	Hits@1	0.8102	**0.8138**
	Hits@10	0.9343	**0.9377**
	Hits@50	0.9735	**0.9752**
	Hits@100	**0.9840**	0.9839
	MAP	0.8546	**0.8586**

Figure 6: Attention vs. GCN Results

From the results, it would seem that the attention model performs similarly to the GCN, so we did not pursue this further. Due to time constraints, we could not test all possible variations, so this is something to look into for future work. However, from the results on our simple attention model (with the same number of layers and hidden dimensions as the GCN model), we concluded that attention did not add anything meaningful and was less efficient than the GCN approach.

5 Discussion and future work

From all these experiments, it seems that our best results are from using the hybrid loss formulation. This makes sense as we are trying to minimize both the loss for predicting accurate alignments and the loss for creating the relational similarity matrix. Clustering, hybrid loss and better representations all pose interesting future directions, which we will briefly discuss.

First, we would like to test on monolingual datasets such as DWY15K, an entity alignment dataset for DBpedia-Wikidata and DBpedia-YAGO. Currently, our data is all cross-lingual (English/Chinese for example), so it would be interesting to see how the alignments from this method compare on monolingual data. In addition, further experiments on other languages such as French and Japanese could be useful for a case analysis on the intended impacts of our proposals.

We may also want to experiment with different clustering mechanisms, or using the similarity matrices differently. Currently we apply a weighted sum to produce a final similarity matrix. However, we could instead explore how using majority rule or other related procedures to decide which entities are similar. In addition, for clustering the common attributes, we can further prune the attributes by another criteria, such as through uncertainty with Bayesian Networks or information gain based on clustering techniques covered in CS 245.

We would also like to test more advanced GCN models. We briefly tested an attention network but there are many variants and different options to choose from. In addition, there exist other graph networks like Relational GCN. Since new GCN models are coming out which may preserve and embed information better than our current version, we believe that this is an important direction track as better embeddings lead to better attribute and entity representations.

We would also like to look at dynamic indices- fast and efficient retrieval requires appropriate index structures that meet these demands. We would like to explore the method of Dynamic Ordered Multi-field Index (DOMI) for information retrieval proposed in [3]. This is especially useful when the knowledge graphs are updating to incorporate new data. Otherwise, the procedure is very inefficient as we will have to repeat large portions of code in order to handle adding new indices. The method uses segmented radix trees and hash tables to improve look-up performance for single or multiple queries. Other approaches we can consider for future work is implementing include B-trees and the Adaptive Index Buffer proposed in [8].

6 Roles of team members

This section documents the role of each team member in this project. From a high-level overview, we ended up splitting up the work after the survey paper/project proposal into: George/Tameez/Sanjeev is in charge of the presentations + content + initial implementations; Yinsheng/Maokai/Fengyuan is in charge of running and creating a pipeline for all the experiments to be tested. In addition, we all equally split the survey paper.

- **George**: Wrote: Project + Final Presentation/Slides, Final Report, Survey paper, project proposal (clustering section); Implementing + experimenting: hybrid loss, Cosine similarity over Sets, clustering common attributes with embeddings. Serving as one of the main points of contact for project related questions.
- **Sanjeev**: Wrote: Project + Final Presentations/Slides, Final Report, Survey paper, project proposal (dynamic indices, datasets); Implementa-

tion + experimenting: supervised loss, clustering common attributes with embeddings

- **Tameez**: Wrote: Project + Final Presentation/Slides, Final Report, Survey paper, project proposal (embedding); Implementation + experimenting: new FC layers in GCN, graph attention network instead of GCN for embeddings, added BERT word embedding, and new word embedding based set similarity
- **Yinsheng**: Survey paper, Running experiments for EMGCN with sigmoid, testing cosine similarity, baseline and BERT on fr_en dataset as well as baseline model performance with different embeddings.
- **Maokai**: Survey paper(EMGCN), Running experiments, including Relu, Leaky Relu, implementing embeddings and exploring DWY15K and DWY100K integration.
- **Fengyuan**: Survey paper, Running experiments with different activation functions, testing Hybrid loss with different ratio, exploring and applying GPU acceleration, Proposing and implementing and exploring fasttext and GCS word embedding.

7 Links

Data download (DBP15k): `https://rb.gy/tt9jsa` (main data used for eval).
Data download (DWY100k): `https://rb.gy/uaiqcs` (no public DWY15K found).
Github: `https://github.com/tluccs/CS245_final_project`

References

[1] Antoine Bordes et al. "Translating Embeddings for Modeling Multi-relational Data". In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: `https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf`.

[2] Muhao Chen et al. *Multilingual Knowledge Graph Embeddings for Cross-lingual Knowledge Alignment*. 2017. arXiv: `1611.03954 [cs.AI]`.

[3] Sura I.Mohammed Ali, Fatma Omara, and Hussien Sharaf. "Information retrieval using dynamic indexing". In: *2014 9th International Conference on Informatics and Systems, INFOS 2014* (Feb. 2015), PDC93–PDC101. DOI: `10.1109/INFOS.2014.7036684`.

[4] Tam Thanh Nguyen et al. "Entity alignment for knowledge graphs with multi-order convolutional networks". In: *IEEE Transactions on Knowledge and Data Engineering* (2020).

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

[6] Zequn Sun et al. “Bootstrapping Entity Alignment with Knowledge Graph Embedding”. In: *IJCAI*. 2018, pp. 4396–4402.

[7] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: `1710.10903 [stat.ML]`.

[8] Hannes Voigt et al. “Adaptive Index Buffer”. In: Apr. 2012, pp. 308–314. DOI: `10.1109/ICDEW.2012.39`.

[9] Zhichun Wang et al. “Cross-lingual Knowledge Graph Alignment via Graph Convolutional Networks”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 349–357. DOI: `10.18653/v1/D18-1032`. URL: `https://aclanthology.org/D18-1032`.

[10] Yuting Wu et al. “Relation-Aware Entity Alignment for Heterogeneous Knowledge Graphs”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. 2019, pp. 5278–5284.