Master Thesis

Active Learning for Entity Alignment

Author: Oshada Nadith Sri Senaweera
Examiner: Prof. Dr. Volker Tresp
Supervisor: Max Berrendorf
Degree programme: Master of Data Science
Ludwig Maximilian University of Munich
February 2020

Abstract

Knowledge Graphs (KGs) represent real-world information or facts in the form of entities and relationships between them. In KGs, facts are represented in the form of "SPO" triples (subject, predicate, object). Popular examples of KGs are Wikidata, YAGO, DBpedia, NELL, Freebase and Google Knowledge Graph. Many of the KGs that have been created are done separately for a particular purpose. However, applications that use KGs have a more diverse knowledge requirement that a single KG cannot satisfy. To tackle this problem, it is essential to integrate multiple KGs into a unified KG, which can satisfy the diverse knowledge requirements of applications. Integration of these heterogeneous KGs can be done via entity alignment, i.e. identification of entities in different KGs that represent the same real-world entity. Manually doing so might ensure high quality, but soon becomes infeasible when confronted with large graphs.

Hence, this study "evaluate different Active Learning Heuristics for the task of entity alignment in KGs" so that the best performing AL heuristics can be identified. To achieve this, instances(sample points) will be picked according to an active learning heuristic and the relationship between performance and the number of instances is studied. AL heuristics implemented in the study are centrality-based and model-based. As per the findings, best performing AL heuristics are based on the centrality where betweenness centrality performs the best.

Keywords: Active Learning, Entity Alignment, Knowledge Graphs

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Specification of Contribution . . . 2
- 1.3 Significance of the Study . . . 2
- 1.4 Outline . . . 3

2 Theory and Definitions **4**
- 2.1 Knowledge Graphs . . . 4
- 2.2 Entity alignment . . . 4
- 2.3 Active learning . . . 5
  - 2.3.1 Use of Active Learning . . . 5
  - 2.3.2 Settings for Active Learning . . . 6
  - 2.3.3 Query Strategies . . . 6
- 2.4 Other Theories and Definitions . . . 10
  - 2.4.1 Loss Function . . . 10
  - 2.4.2 Similarity Measures . . . 10
  - 2.4.3 Evaluation Metrics . . . 11
  - 2.4.4 Early Stopping . . . 11
  - 2.4.5 Centrality Measures . . . 11

3 Related Work **14**
- 3.1 KG Embedding . . . 14
- 3.2 Embedding-based KG Alignment . . . 15
  - 3.2.1 JE . . . 15
  - 3.2.2 MTransE . . . 15
  - 3.2.3 JAPE . . . 16
  - 3.2.4 GCNAlign . . . 17
  - 3.2.5 Comparison Between Alignment Models . . . 19
- 3.3 Active learning for Knowledge Graph Embedding . . . 19

4 Settings and Implementation **21**
- 4.1 Experimental Setup . . . 21
- 4.2 Datasets . . . 21
  - 4.2.1 DBP15K . . . 22
  - 4.2.2 WK3l-15K . . . 23
- 4.3 Implementation . . . 24
  - 4.3.1 AL Framework Implementation . . . 24

4.3.2 GCNAlign Implementation 27
4.3.3 Performance Evaluation 27

5 Analysis and Results **28**
5.1 Performance of AL Heuristics with Number of Queries 28
5.2 Performance of AL Heuristics with the Training Data Size 33
5.3 Comparison of Performance with Number of Queries and Percentage of Data Used . 36
5.4 Performance and Data-sets 37

6 General Discussion and Conclusions **38**
6.1 Overview of the study 38
6.2 Discussion of Implementation Steps 39
6.3 Summary of study findings 40
6.4 Discussion of the results 40
6.5 Limitations 40
6.6 Future Implementations 41
6.7 Conclusions 41

Chapter 1

Introduction

Recently, the use of knowledge graphs(KGs) has become wider and diverse. For example, KGs are used extensively in areas such as search engines, artificial intelligence-related applications, Natural language processing, etc. The main aim of KGs is to organize human knowledge in a machine-readable format so that it can be accessed and used easily. Therefore, KGs can be used as an important infrastructure facilities for many of these above applications. KGs are dynamic in the sense that they can be easily extended to incorporate new data, which will enable KGs to re-purpose and provide new insights and inferences. These qualities distinguish KGs from regular databases which get populated and ultimately become dormant. Therefore, the use of KGs in the above applications makes them more efficient and effective. In this respect, many research fields related to KGs have been established and a considerable number of research papers are published in recent years.

1.1 Motivation

As discussed earlier, KGs have become widely used to organize information. Popular examples of KGs are Wikidata[1], DBpedia[2], NELL [1], Freebase [2], YAGO [3] and Google Knowledge Graph[3]. These KGs conatin millions of facts. We can define a knowledge graph as a semantically structured collection of facts. In KGs, facts are represented in the form of "SPO" triples. SPO triple can be defined as below;

$$(Subject, Predicate/Relationship, Object)$$

So the existence of a SPO triple indicates a particular fact in the KG.

To construct a KG, we combine all SPO triples to form a multi-graph where nodes represent entities(all subjects and objects) and directed edges represent relations. The direction of the edge is subject to an object. Also, we give a type or label to the edge to represent different relation types. This formation of data can be identified as relational data. Often, these existing KGs miss many facts and some of the edges they contain are incorrect. Hence statistical Relational learning (SRL) is applied for these KG relational data to get relational models. SRL is the creation of statistical models for relational data. These SRL models of KGs will enable tasks such as KG completion(identifying missing edges), entity resolution and link-based clustering [4].

[1] `www.wikidata.org`
[2] `https://wiki.dbpedia.org`
[3] `https://developers.google.com/knowledge-graph`

Many of the KGs that have been created are done separately for a particular purpose, therefore most KG models focus on modeling single KG. Moreover, most of these KGs are heterogeneous, such as they are multilingual. However, the application that uses KGs has a more diverse knowledge requirement that a single KG cannot satisfy. To tackle this problem, it is essential to integrate multiple KGs into a unified KG, which can satisfy the diverse knowledge requirements of applications. The integration of these heterogeneous KGs can be done via entity alignment method. That is the identification of entities in different KGs that denote the same real-world entity. In recent times, there is an uptick on the research published in this regard. For example, entity alignment techniques such as JE [5], MTransE [6], JAPE [7] and GCNAlign [8] have been published. These methods mostly are based on the embedding of entities and relations of two KGs into a common embedding space. These methods will be discussed in detail in the Chapters 3. However, the main drawback of these methods is that these methods rely heavily on existing entity alignment as training data. In practical real-world KGs, the accessible prior alignment between two KGs is small compared to the size of the KGs. This limited training data will prevent learning accurate embedding from these models. To tackle this problem of limited pre-alignment data, we can manually align entities of two KGs and use them as training data. But manually aligning would be very complicated and expensive. In this regard, we can use Active Learning (AL), which would enable us to obtain training data that would maximize the performance of the above methods with limited training data. In this context, the motivation of the study is to explore the application of active learning techniques to the task of entity alignment.

1.2 Specification of Contribution

The main objective of the study is to "evaluate different Active Learning Heuristics for the task of entity alignment in KGs" so that the best performing AL heuristics can be identified. To achieve this, instances(sample points) will be picked according to an active learning heuristic and analyze the relationship between performance with the number of instances. As per the secondary objectives, the following will be studied with respect to the use of AL for entity alignment task:

- Compare different active learning heuristics under different settings (e.g - different data-sets, different performance measures).
- Compare the performance of heuristics and Identify the best performing AL heuristic for different circumstances.
- Identify the factors that the performance of heuristic depends on.

1.3 Significance of the Study

The use of Active learning in machine learning or other related fields are reasonably limited. In a survey conducted among researchers, who do annotation for natural language processing [9], only 20% of the respondents mentioned they had ever decided to use active learning. This shows the skepticism among researchers about the practical usefulness of AL. When consider the entity alignment task, AL is not used at all (per my knowledge) for it. As mentioned above, AL can be a major solution for the main problem most KG alignment models currently have, lack of prior alignment data for training. This setting is perfect for AL as it is best suited when there is limited labeled data and abundant unlabelled data. Therefore this study will focus on studying the use of AL techniques for the task of entity alignment. The findings of the study will help understand whether AL can be

successfully used for KG alignment. Also, the study will be useful for further studies related to KG alignment to use as a basis.

1.4 Outline

The basic overviews of the chapters that are included in the thesis can be outlined as follows. Chapter 1 gives a brief introduction to the study area and highlights the significance and objectives of the study. Chapter 2 consists of details about theories and definitions related to the thesis. It gives a basic overview of the theories and definitions used in the study, which will help better understand the methods and techniques used in the study. Chapter 3 focuses on the published literature regarding the study area. This mainly includes studies that focused on "entity alignment of knowledge graphs" and "active learning". Chapter 4 gives an overview of the settings such as setup foe the study and also gives how the experiments were implemented to achieve the objectives. Chapter 5 analyze the data gathered from experiments and give findings from the analysis. In Chapter 6, a general discussion about the experiments, analysis, results will be given. Further, Chapter six will explore shortcomings and suggest future improvements to the study. Finally, it will give conclusions from the study.

Chapter 2

Theory and Definitions

This chapter provides information about the most important theories and definitions used throughout the study.

2.1 Knowledge Graphs

Knowledge graphs represent real-world information or facts in the form of entities and relationships between them. KG typically consists of a set of entities, a set of relations and a set of triples. There exist some KGs with attribute information. However we do not use attribute information, therefore we will focus on defining KG without considering it.

Entity of a KG can be considered as a real-world object with a unique id. Relation describes connections between these entities. Triples consist of entities and relations which represent real-world fact. These triples are referred as relational triples and we can represent relational triples in the form $(entity_1, relation, entity_2)$. For example, in YAGO $Albert\_Einstein$ and ETH_Zurich are entities and $graduatedFrom$ is a relation. Then $(Albert\_Einstein, graduatedFrom, ETH\_Zurich)$, is a relational triple. Now let denote the sets of entities and relations E,R respectively. Also let $T_R \subset E \times R \times E$ be a set of relational triples. Now we can represent a KG as $G = (E, R, T_R)$, which we will use throughout the report.

2.2 Entity alignment

Entity Alignment refers to the matching of entities in different KGs with each other, which correspond to similar objects in the real world. There are two commonly used approaches for entity alignment; string-similarity based entity alignment and embedding based entity alignment. String-based aligned models use string similarity of entities as the main tool for alignment. For examples, alignment models LIMES [10], RDF-AI [11] and SILK [12] use the this approach. These methods use the string value of entity labels to do a comparison between entities. For example, LIMES first calculates the approximate similarity between entities using string labels and the triangle inequality. Then, the actual similarity of entity pairs with high approximate similarity is calculated and pairs with the highest actual similarity are returned. When KG alignment is cross-lingual these approach can become a little more cumbersome. In this situation, a method like multi-lingual BERT [13] to compare strings across different languages will be needed.

In recent time, KG embedding models has been successfully used to model KGs. This has lead to the introduction of embedding-based models for entity alignment task. Most of these embedding based alignment models are built on top of a KG embedding model such as TransE [14]. These models are defined assuming alignment between two KGs, but can be extended to aligning of multiple KGs. Further, models primarily focus on the structures of the KGs when aligning. We will discuss the KG embedding and several embedding based KG alignments models in Chapter 3 including GCNAlign [8], the model used in the study.

2.3 Active learning

Active learning(AL) is a sub-field of artificial intelligence and machine learning, where the hallmark of an active learning system is that it eagerly develops and tests new hypotheses as part of a continuing, interactive learning process [9]. Simply put, active learning will rank/choose instances according to their importance. The importance can be defined as providing the most information to the machine learning model or algorithm. For example, consider a binary classification task where we learn a classifier (ex-half-space) by training on some training data. The instances that would provide most information here would be the instances that are situated near the Bayes optimal classifier. Therefore, active learner develops a "line of inquiry" same way a scientist design a series of experiments. Hence active learning is sometimes called "query learning" in the computational learning theory literature and is closely related to work in "optimal experimental design" or "sequential design" in statistics.

An active learning setting is based on situations where we want to train a model with some training data, but the training data contain no labels for the training instances. In statistical modeling, this corresponds to no values for the dependent variable (mostly notated by Y). Therefore the AL framework consists primarily of two components. They are;

- **Query System** which is used to query the instances from the training data which contain no label.
- **Oracle** which will label the instances that are picked by the query system.

Among the above two components, most work on AL is focused on query system, where multiple query strategies have been studied and researched with respect to various aspects of machine learning techniques.

2.3.1 Use of Active Learning

As mentioned above, query selection algorithms are the most focused area in AL. However, before diving into that section, it is important to identify scenarios in which AL is not appropriate.

Using AL in situations where instance labels come at little or no cost will not be appropriate if AL implementation would create extra engineering overhead and not very cheap when labels can be easily obtained with little to no cost. Another situation AL is inappropriate would be when we can train a model accurately with a small number of randomly obtained label instances. The use of AL framework in this situation might be more expensive than just collect the small number of labeled instances that are required.

Therefore, AL is most appropriate when numerous (unlabeled) data instances that can be easily collected or synthesized is available, and we expect our model needs a lot of labeled instances (or few appropriately labeled instances) as training data to be properly trained. Further, it is also assumed that the oracle gives the correct labels for the instance queries, and the hypothesis class

(examples naive Bayes, decision trees, neural networks, etc.) that is going to be used for the problem is pre-decided (for AL strategies that are model-based).

2.3.2 Settings for Active Learning

Given that active learning is appropriate, there are some common settings in which the learner will query the labels of instances. The three main settings that appear in the literature are:

- **Query Synthesis:** In this setting, the learner generates an instance (from some underlying natural distribution) within input space and query the label of that instance.
- **Stream-based Selective Sampling:** In this setting, it is assumed that obtaining an unlabeled instance is free (or inexpensive). Then each unlabelled instance is drawn one at a time, and the learner determines whether to query the label of the instance or reject it. There are several ways to decide whether to query or discard an instance. One approach is to use the informativeness of the instance where instances with higher informativeness are more likely to be queried. Another approach is to compute a region of the instance space that is ambiguous (region of uncertainty) to the learner, and only query instances that fall within it. These approaches will be discussed in detail under query strategies.
- **Pool-based Sampling:**This setting assumes that there is a large pool of unlabeled data (U) available with a small set of labeled data (L). Instances are then drawn from the pool which is usually assumed to be closed (not-changing), although this is not strictly necessary. Queries are chosen according to a utility measure (informative measure) which is applied to all instances in the pool (if U is very large, a sub-sample of the pool). Typically queries with the highest informativeness are selected.

When consider the setting Query Synthesis, sometimes it can be problematic. This is because data-generating distribution is not taken into account (and may not even be known) when generating an instance to query. Therefore active learner runs the risk of querying arbitrary instances devoid of meaning. This limitation is avoided in stream-based and pool-based scenarios. The main difference between stream-based and pool-based active learning is that stream-based AL sequentially draws one instance at a time and make query decision of each instance individually, whereas in pool-based active learning all unlabelled data is evaluated and ranked and then the best queries are selected.

When consider the applicability among these settings, the most applicable setting is Pool-based Sampling. For many real-world learning problems, a large collection of unlabelled data can be gathered at once which is the setting that pool-based AL is based on. However, there are some settings where the stream-based approach is more appropriate. For example, when the data is large and cannot load to memory and need to be scanned sequentially from disk. When consider the setting of the study, like most real-world situation our AL framework is also based on pool-based scenario.

2.3.3 Query Strategies

The main difference between active and passive learning is, in active learning, learners actively query instance to learn from pool-of learning data whereas, in passive learning, the learner is provided with a set of data to learn. As mentioned earlier, the main area in AL is the querying of the instances. To query the instances, we need an informativeness measure of the unlabelled instances. All the query strategies focus on this informativeness measure. Below we will discuss some main Query strategies in AL literature. There does not exist an optimal query strategy, rather the best strategy depends

on the specific circumstances of the AL framework. Main areas of query strategies in AL literature are;

1. Uncertainty Sampling
2. Reducing the hypothesis space
3. Minimizing expected Error and Variance
4. Exploiting structure in the data

2.3.3.1 Uncertainty Sampling

The idea here is that the learner will query the instance that it is least confidence about or avoid querying the instances that it is already confident. In other words, query the instances that the learner is most uncertain about the decision of the label that should be given. So in uncertainty sampling, we measure this decision uncertainty of each instance and select the instances with the highest uncertainty.

To put this idea in a probabilistic perspective, we need a probabilistic classifier that will output a posterior probability distribution $P_\theta(Y|x)$ over the label variable Y given the input x and learned model parameters θ. Under this interpretation, what we want is to query the instance for which $P_\theta(\hat{y}|x)$ is closest to a uniform distribution (here $\hat{y}$ refers to the classifier's most likely prediction for x). For example, in the case of binary classification, query the instance for which $P_\theta(\hat{y}|x)$ is close to 0.5. However, when the model has posterior distributions over three or more labels we need a more general measure of uncertainty or information content. Below given are the three main uncertainty measures. We will denote the best instance that the utility measure A would select for querying as x^*_A.

- **Least Confident:** Query the instance for which the learner has the least confidence in the predicted output. This can be defined as below;

$$\begin{aligned} x^*_{LC} &= \underset{x}{\text{argmin}}\ P_\theta(\hat{y}|x) \\ &= \underset{x}{\text{argmax}}\ 1 - P_\theta(\hat{y}|x) \end{aligned} \tag{2.1}$$

  where $\hat{y} = argmax_y\ P_\theta(y|x)$, the prediction with the highest posterior probability under the model θ.

- **Margin:** Learner select the instance that has the smallest difference between the first and second most probable labels, which is defined as below;

$$\begin{aligned} x^*_M &= \underset{x}{\text{argmin}}\ [P_\theta(\hat{y_1}|x) - P_\theta(\hat{y_2}|x)] \\ &= \underset{x}{\text{argmax}}\ [P_\theta(\hat{y_2}|x) - P_\theta(\hat{y_1}|x)] \end{aligned} \tag{2.2}$$

  where $\hat{y_1}$ and $\hat{y_2}$ are the first and second most likely predictions under the model, respectively.

- **Entropy:** The entropy(denoted by H) formula is applied to each instance. Instance which has the highest entropy is then queried. Entropy is defined as below;

$$\begin{aligned} x^*_H &= \underset{x}{\text{argmax}}\ H_\theta(Y|x) \\ &= \underset{x}{\text{argmax}}\ -\sum_y P_\theta(y|x)\ logP_\theta(y|x) \end{aligned} \tag{2.3}$$

where y ranges over all possible labeling of x.

When consider the least confident measure, it only considers information about the best prediction. Hence it does not use the rest of the information of the posterior distribution. This shortcoming is addressed in Margin sampling by incorporating the second-best labeling in its assessment. However, in multi-label models when the number of labels is large, the margin approach still ignores much of the output distribution. So to utilize all the possible label probabilities, we can use entropy sampling. Therefore entropy is the most commonly used utility measure in uncertainty sampling. In our study, we will also use the entropy measure in our uncertainty sampling implementation.

2.3.3.2 Reducing the Hypothesis Space/Version Space

In machine learning, a "hypothesis" is identified as a particular computational model that learns its parameters through training data and make predictions on new data. Hypothesis space which is denoted by H is the set of all hypotheses under consideration. For example, if we consider a perceptron, then all possible weights for parameters in a perceptron could be considered as H. Now let's consider version space, which is denoted by $V \subseteq H$. Version space [15] is defined as a subset of hypotheses that are consistent with the training data. That is, hypotheses in version space make the correct decision for all labeled training instances (assume we already have some labeled data). If we assume that there is a hypothesis that can perfectly explain the data(true hypotheses), then as we collect more labeled data, the area of the version space $|V|$ should become smaller. That is, a set of hypotheses in V will approximate the true hypotheses more accurately. This suggests that an active learning algorithm should try to obtain new training instances which will quickly minimize $|V|$. Two active learning algorithms explicitly motivated by reducing the version space are;

- **Query By Disagreement(QBD):** QBD assumes the stream-based selective sampling scenario (see Section 1.3), where data come in a stream and the learner decides whether to query or discard them in real-time. The approach essentially maintains the working version space V, and if a new data instance x comes along for which any two legal hypotheses disagree, then x's labeling cannot be inferred and its true label should be queried. However, if all the legal hypotheses do agree, then the label can be inferred and x can be safely ignored.

- **Query By Committee(QBC):** Any disagreement based approach which uses a "committee" is referred to as QBC. Here committee is an ensemble of hypotheses, where we can use general ensemble algorithms such as bagging or boosting to construct a committee. So to implement this, what is required is a method for obtaining hypotheses in the committee and a heuristic for measuring disagreement among them. There are several ways for measuring disagreement in classification tasks, but we will focus on two dominant trends. One is vote entropy, which is defined as:

$$x^*_{VE} = \underset{x}{\text{argmax}} - \sum_y \frac{vote_C(y,x)}{|C|} \; log \frac{vote_C(y,x)}{|C|} \tag{2.4}$$

  where y ranges over all possible labelings, $vote_C(y,x) = \sum_{\theta \in C} \mathbb{1}_{h_\theta(x)=y}$ is the number of "votes" that the label y receives for x among the hypotheses in committee C, and |C| is the committee size.

  Other disagreement measure is based on Kullback-Leibler (KL) divergence [16], which is an information-theoretic measure of the difference between two probability distributions. In this case, we want to quantify disagreement as the average divergence of each committee member

θ's prediction from that of the consensus C.

$$x^*_{KL} = \underset{x}{\text{argmax}} \frac{1}{|C|} \sum_{\theta \in C} KL(P_\theta(Y|x) \;||\; P_C(Y|x)) \tag{2.5}$$

where $KL(P_\theta(Y|x) \;||\; P_C(Y|x)) = \sum_y P_\theta(y|x) log \frac{P_\theta(y|x)}{P_C(y|x)}$

However, we do not have any implementation of this version space reduction in our study. This is because what we deal in our study is graph data and it would be hard to do a implementation of AL framework using this query strategy.

2.3.3.3 Minimizing Expected Error and Variance

Here, we want a learner to choose instances that, once it knows the label, and trained with those instances, the learner will most likely have the minimum error or variance that can be achieved in its prediction. To compute the expected error, we need two probability distributions: 1) the probability of the oracle's label y in answer to query x; and 2) the probability that the learner will make an error on some other instance x', once the answer is known for query x. However, neither of these probabilities are known. But for both these cases, the model's posterior distribution can be used as a reasonable approximation. If we have a large unlabeled pool U available, the learner can attempt to minimize the expected error over it, assuming that it is representative of the test distribution. For example, consider we want to minimize the expected classification error (or 0/1-loss) over the unlabeled data U.

$$\begin{aligned} x^*_{ER} &= \underset{x}{\text{argmin}}\; E_{Y|\theta,x}\Big[\sum_{x' \in U} E_{Y|\theta^+,x'}[y \neq \hat{y}] \Big] \\ &= \underset{x}{\text{argmin}} \sum_y P_\theta(y|x) \Big[\sum_{x' \in U} 1 - P_{\theta^+}(\hat{y}|x') \Big] \end{aligned} \tag{2.6}$$

where θ^+refers to the a new model after it has been re-trained using a new labeled set $L \cup (x, y)$, that is, after adding the candidate query x and the hypothetical oracle response y.

Here, the use of 0-1 loss function means that the objective is to minimize the expected total number of wrong predictions, not giving any credit to near misses. More flexibility can be obtained by using some other loss function such as log-loss.

According to the paper [17] expected error can be decomposed to noise, bias, and variance. Learner cannot do anything about the bias or noise, hence only option to reduce the expected error is to reduce the variance. Therefore, minimizing variance guarantees that models' future generalization error will be minimized. Therefore, what we want is to pick the instances that are expected to most reduce the model's output variance over the unlabeled instances U :

$$x^*_{VR} = \underset{x}{\text{argmin}} \sum_{x' \in U} Var_{\theta^+}(Y|x') \tag{2.7}$$

When we consider this strategy to the our situation, it is would be hard to have an implementation of this strategy in our AL framework. Hence we ignore this strategy in our study.

2.3.3.4 Exploiting Structure in the Data

1) Density-weighted Heuristics

The basic idea is that informativeness of an instance not only based on information content but also on the representative of the underlying distribution in some sense (e.g., inhabit dense regions of the

input space). The generic information density heuristic captures the main ideas of this approach:

$$x^*_{ID} = \underset{x}{\operatorname{argmax}}\ \phi_A(x) \times \left(\frac{1}{U} \sum_{x' \in U} sim(x, x') \right)^{\beta} \tag{2.8}$$

Here, U is the size of the unlabeled pool and $\phi_A(.)$ represents the utility of x according to some other query strategy A, such as uncertainty sampling,QBC etc. The second term calculate the representatives of x in terms of the density. Parameter β controls the relative importance of the density term. Representatives of x is calculated by getting its average similarity to all other instances in the input distribution, where similarity is calculate using some similarity measure(e.g., cosine similarity).

2) Cluster-Based heuristics

Another Exploiting structure method is Cluster-Based AL. Some methods in this regards are; pre-cluster the data and begin by querying cluster centroids to comprise the initial label data, rather than starting with a random sample [18], and another is to cluster the most informative instances after each iteration and query the most representative instances of those clusters [19]. However, these methods may have some problems which are given below.

- There may be no obvious clustering of or a good similarity measure for clustering the data is unknown.
- Difficulty of identifying a suitable number of clusters.
- Clusters may not correspond to the hidden class labels.

Even-though with these problems cluster-based AL can be used to take advantage of the data structure when they happen to be informative.
Since in our study, we work with graph data, exploiting the structure of the data would be the easiest and most cost-efficient way to implement AL frameworks. Hence most of the query strategies we implement in our study are based on the concept of exploiting the structure in the data.

2.4 Other Theories and Definitions

Below gives some other basic theories and definitions used in the study, which will help understand the study more clearly.

2.4.1 Loss Function

We use the Pair-wise Margin Loss function. Here, the margin loss for each given pair is calculated and the mean of it is taken. Here the pair refer to node pair, which consists of a node from each KG. Margin-based loss function focus on how well the prediction agrees with the sign of the target, not on the difference between true target and prediction. In our case, pairs and margin are defined the same as given in GCNAlign loss function.

2.4.2 Similarity Measures

There are multiple similarity measures exist to calculate similarity between two objects. These measures use the embeddings derived for the objects (our case objects are entities) to calculate the

similarity. In the study, we use L1-similarity which is based on L1-distance, to calculate the similarity. Let X and Y be two embeddings with embedding dimension of n, where $X = (x_1, x_2, ..x_n)$ and $Y = (y_1, y_2, ..y_n)$. Then, L1-distance is defined as follows; $L1(X,Y) = \|X - Y\|_1 = |x_1 - y_1| + ...|x_n - y_n|$. Then the L1-similarity of X and Y, $L1Sim(X,Y)$ is defined as below;

$$L1Sim(X,Y) = \frac{1}{1 + L1(X,Y)} \tag{2.9}$$

Values for L1-similarity measure defined above fall in the boundary [0,1] where values close to 1 indicate higher similarity and values close to 0 indicate lower similarity.

2.4.3 Evaluation Metrics

To evaluate the performance in embedding based KG alignment models, typically used metrics are Hit@k, Mean Rank (MR) and Mean Reciprocal Rank (MRR). To define these measures, assume we have two sets of entities from two KGs we want to align. These two sets represent alignment data where for each entity in a set, there is a corresponding align entity (similar entity) in other set. Now consider an entity in a one set, calculate similarities between that entity and all other candidate entities in other set, and rank those according to similarity (higher the similarity lower the rank). Repeat this to all entities in that set. Now we can compare these ranks with the rank of actual align entity. From this rank, the above measures can be defined as follows;

- **Hits@k:** The proportion of the correct entities ranked in the top k. Normally used k values are 1/10/50/100 (Hits@1, Hits@10, Hits@50, Hits@100).
- **Mean Rank:** Average rank of the correct entity.
- **Mean Reciprocal Rank:** Mean of the reciprocal rank. Reciprocal rank is defined as $\frac{1}{Rank}$ where rank is the position of the correct entity.

2.4.4 Early Stopping

Early stopping is a regularization technique that is used to stop models from over-fitting on training data. The approach is that set the model to train for a large number of training epochs. In the training, after every k epoch (k=1,2,... and can be changed depending on the situation) model is evaluated on a holdout validation data-set. If the model performance on the validation data starts to decrease (or does not increase), then the training is stopped and the model at that time is chosen. Normally, the first decrease in performance is not the best time to stop training. That is because the model may in a phase of slight performance decrease before getting much better. To avoid this, we can add a delay trigger in terms of the number of epochs with decreases (or no improvements) after which training will be stopped. This is known as “patience” argument in Early stopping. Model performance decrease would refer to things like increase in loss or decrease in accuracy. In our case, we use "Hits@1" and once its value on the validation set starts decreasing we stop the training.

2.4.5 Centrality Measures

Graph Centrality is a concept used to identify the importance of a node in a graph. Here the importance of a node can be considered as to how central a node is in the graph. This importance of a node depends on the angle it is defined. Here, angle refers to characteristics of the nodes such as connectivity, reachability, etc. There are various methods to measure the centrality of a node, from

classic methods (e.g. degree centrality, closeness centrality) to recent eigen-vector based methods (e.g. page rank centrality). Below we will define centrality measures used in the study.

Degree Centrality

The degree centrality for node e is the fraction of nodes that it is connected to. That is, degree centrality of node e, $C_D(e)$;

$$C_D(e) = \frac{degree(e)}{N-1} \tag{2.10}$$

where $degree(e)$ is the number of other nodes that node e is connected and N is the total number of nodes in graph.

Closeness Centrality

Closeness centrality, $C_C(e)$ of node e is the reciprocal of the average shortest path distance to node e over all n-1 reachable nodes, which is defined as;

$$C_C(e) = \frac{n-1}{\sum_{v \neq e} d(v,e)} \tag{2.11}$$

where $d(v,e)$ is the shortest-path distance between v and e, and n is the number of nodes that can reach node e.

Betweenness Centrality

Betweenness centrality of node e is the sum of the fraction of all-pairs shortest paths that pass through node e. Let $C_B(e)$ denote the betweenness centrality, then it can be defined as;

$$C_B(e) = \sum_{s,t \in E} \frac{\sigma(s,t|e)}{\sigma(s,t)} \tag{2.12}$$

where E is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t)-paths, and $\sigma(s,t|e)$ is the number of those paths passing through some node e other than s,t. If $s = t$, $\sigma(s,t) = 1$, and if $e \in (s,t)$, $\sigma(s,t|v) = 0$.

Harmonic Centrality

Harmonic centrality of a node e is the sum of the reciprocal of the shortest path distances from all other nodes to node e. This is defined as;

$$C_H(e) = \sum_{v \neq e} \frac{1}{d(v,e)} \tag{2.13}$$

where $d(v,e)$ is the shortest-path distance between nodes v and e.

Page Rank Centrality

Page Rank ranks the nodes in a graph based on the structure of the incoming links. To calculate page rank, two components are required; Transition matrix, and initial importance vector. Let there be N nodes in the graph. Then dimensions of transition matrix and initial importance vector is $N \times N$ and $N \times 1$. Let node e has d_e out links, then element $M_{ve} = \frac{1}{d_e}$ if $e \to v$ else $M_{ve} = 0$. Page rank is defined as the solution of $R = MR$ where R is page rank importance vector. To derive R, we solve the equation using power iteration by using the initial importance vector r. That is $R = M^k r$ where we increase value of $k = 1, 2, 3, ...$ until solution to R converge. Hence, the final values in R are the Page Rank values of the nodes in the graph.

For all these centrality measures considered, higher values indicate higher centrality and hence the nodes with higher centrality will have a higher importance in the graph.

Chapter 3

Related Work

This chapter provides a brief description and comparison of studies that have been done regarding Knowledge Graph Alignment. Also, we will discuss studies that have used active learning techniques with respect to Graphs. Proper study of published literature on the study area will help identify areas that study should focus on, and provide a road map on how to conduct the study.

3.1 KG Embedding

In recent years, many works are done in the area of KG embedding. KG embedding models give embeddings of entities and relations of KG. Here, embeddings are either a function from the entities/relation to a lower-dimensional vector space or it is the result, i.e. a vector. These embedding then can be used to relationship prediction(KG completion), information extraction and other tasks. The most prominent KG embedding model is the TransE [14].

Given a relationship triple (h, r, t), TransE suggest that embedding of the tail entity t should be close to the embedding of the head entity h plus the embedding of the relationship r (predicate), $h + r \approx t$. Let S be the training set of triples (h, r, t) composed of two entities $h, t \in E$ (the set of entities in KG) and a relationship $r \in R$ (the set of relationships in KG). When consider embeddings of these, we use boldface characters (used throughout the Chapter).

The model learn the embeddings by minimizing the margin-based objective function L;

$$L = \sum_{(h,r,t)\in S} \sum_{(h',r,t')\in S'_{(h,r,t)}} [\gamma + d(\mathbf{h} + \mathbf{r}, \mathbf{t}) - d(\mathbf{h}' + \mathbf{r}, \mathbf{t}')]_+ \tag{3.1}$$

where $[x]_+$ denotes positive part of x, $\gamma > 0$ is a margin hyper-parameter, $d(\mathbf{h} + \mathbf{r}, \mathbf{t})$ energy of triplet for some dissimilarity measure d (normally L_1 or L_2 norm), and $S'_{(h,r,t)} = \{(h', r, t)|h' \in E\} \cup \{(h, r, t')|t' \in E\}$ is the set of corrupted triples. Corrupted triples(negative triples) are created by replacing the head or tail of a valid triple by an random entity.

Loss function can be trivially minimized by artificially increasing the embedding norms and shaping the embeddings. To stop this, there is an additional constraint, that the L2-norm of the embeddings of the entities is 1 (no norm constraint on relation embeddings), which will stop the trivial minimization of the loss function in the training process.

TransE is a representative KG embedding approach that tries to get similar embeddings for similar entities. That is to capture the semantic similarity between entities in embedding space. For

example, if a KG includes two entities such as Germany and France, the embedding of Germany should be close to the embedding of France. This is also called structure embedding.

TransE model is simple but powerful and gives good results on link prediction and triple classification problems. In recent years, several studies improving the TransE model have been published; which include TransR (Lin et al., 2015), TransH (Wang et al., 2014) and TransD (Ji et al., 2015). These approaches introduce a new representation of relational translation. There are many other KG embedding approaches, which is given in the survey paper (Wang et al., 2017) with comparisons among each method.

3.2 Embedding-based KG Alignment

In the study we use GCNAlign which is an embedding based KG alignment model. There are other embeddings based KG Alignment methods also. We will discuss and compare these models below.

3.2.1 JE

JE [5] , jointly learns the embeddings of multiple KGs in a uniform vector space to align entities in KGs. For this, JE uses a set of seed entity alignments. To learn the embeddings, JE minimizes a margin-based loss function, which is given below;

$$L = \sum_{(h,r,t)\in S} \sum_{(h',r,t')\in S'_{(h,r,t)}} [\gamma + d(\mathbf{h}+\mathbf{r},\mathbf{t}) - d(\mathbf{h}'+\mathbf{r},\mathbf{t}')]_+ + \lambda_1 \sum_{y\in(h,h',r,t,t')} |\|y\|_2 - 1| + \lambda_2 \sum_{(e_i,e'_i)\in A} \|\mathbf{e}_i - \mathbf{e}'_i\|_2 \tag{3.2}$$

Here, first term is exactly the same as TransE loss fuction, which was defined above. In other two terms, λ_1 and λ_2 are ratio hyper-parameters, and A is the selected seed alignments whose entities are represented by e_i in KG_1 and e'_i in $KG2$. Second term (the λ_1 part) is to stop the trivial minimization of loss function (the λ_1 part) in the training process. Third term is the entitity alignment loss term (the λ_2 part). Addition in JE compared to TransE is the entity alignment loss term which is used to learn the alignment information between KGs. Therefore, the core of the JE is to align two KGs using embeddings in a uniform space that jointly learned via the overlapping parts between the two KGs.

3.2.2 MTransE

MTransE [6], consists of two components that learn on the two facets of KG; the knowledge model that encodes the entities and relations of each KG, and the alignment model that learns the similar entities between different KGs from the existing alignment. In the knowledge model part, TransE is applied for each KGs. So for this part, the loss function is similar to the TransE loss function. Let G be a set of KGs and G_{KG} is a specific KG in G. T=(h,r,t) denotes a triple where h,t are entities and r is relations of a KG. Assuming that aliment is between two KGs (KG_1 and KG_2). Then loss function for the knowledge model is given below:

$$S_K = \sum_{G\in(KG_1,KG_2)} \sum_{(h,r,t)\in G_{KG}} \|h + r - t\| \tag{3.3}$$

The alignment model part constructs the transitions between the vector spaces of KGs. And the alignment loss function is defined as below;

$$S_A = \sum_{(T,T') \in \delta(KG_1, KG_2)} S_a(T, T') \tag{3.4}$$

Here $\delta(KG_1, KG_2)$ refers to the alignment set which contains the pairs of triples that have already been aligned between KG_1 and KG_2. Further $S_a(T, T')$ denotes the alignment score obtained by iterating through all pairs of aligned triples. The align score between a pair can be calculated in three ways; distance-based axis calibration, translation vectors, and linear transformations. Each of them is based on a different assumption, and constitutes different forms of S_a alongside.

Combining the above knowledge and alignment models, MTransE minimizes the following loss function $L = S_K + \alpha S_A$, where α is a hyper-parameter that weights S_K and S_A.

3.2.3 JAPE

JAPE [7], uses both structure embedding and attribute embedding to match entities in different KGs, where it jointly embeds the structures of two KGs into a unified vector space and further refines it by leveraging attribute correlations in the KGs. JAPE employs two models, structure embedding (SE) and attribute embedding (AE), to learn embeddings based on two facets of knowledge (relationship triples and attribute triples) in two KGs, respectively.

SE models the geometric structures of two KGs using existing alignments given beforehand as a bridge to overlap their structures. For this TransE model is implemented, which learns vector representations of entities in the overlay graph of two KGs. Let $tr = (h, r, t)$ be a triple and the plausibility of tr is measured by the score function $f(tr) = \|h + r - t\|_2^2$. Then the SE minimize the following loss function;

$$L_{SE} = \sum_{tr \in T} \sum_{tr' \in T'_{tr}} (f(tr) - \alpha f(tr')) \tag{3.5}$$

Where tr' is the negative triples (defined earlier), α is a ratio hyper-parameter and T and T'_{tr} are set of all positive and negative triples respectively. It is important to make sure that each pair in the seed alignment share the same embedding during training, to bridge two KGs. Note that Eq 3.5 is different from Eq 3.1. Eq 3.1 aims at distinguishing positive and negative triples, and expects that their scores can be separated by a large margin. However, for the entity alignment task, in addition to the large margin between their scores, it is required to assign lower scores to positive triples and higher scores to negative triples. This will reduce the drift of embeddings in the unified space and better capture the common semantics of two KGs. Therefore, Eq 3.5 is used here.

In attribute embedding (AE), attributes are embedded in a way that it captures the correlations of attributes. Here, attributes are considered to be correlated if they are commonly used together to describe an entity. So AE minimizes the following objective function:

$$L_{AE} = \sum_{(a,c) \in H} W_{a,c} \; logP(c|a) \tag{3.6}$$

Where H denotes the set of positive (a, c) pairs, i.e., c is actually a correlated attribute of a, and the term $P(c|a)$ denotes the probability. $W_{a,c} = 1$ if a and c have different range types, otherwise $W_{a,c} = 2$ which increases the probability of embeddings be similar. To establish correlations between attributes of aligned entities, seed entity pairs are used. So, given an aligned entity pair $(e^{(1)}, e^{(2)})$, then attributes of $e^{(1)}$ is correlated to each attribute of $e^{(2)}$, and vice versa. This is added to H

when deriving attribute embedding. The above Eq 3.6 can be modify to include negative samples also, for that term $logP(c|a)$ is replaced with the below term;

$$log\sigma(\mathbf{a}.\mathbf{c}) + \sum_{(a,c')\in H'_a} log\sigma(-\mathbf{a}.\mathbf{c}') \tag{3.7}$$

Where $\sigma(x) = \frac{1}{1+e^{-x}}$,$H'_a$ is the set of negative pairs for attribute a generated according to a log-uniform base distribution, assuming that they are all incorrect, and **a** and **c** are attribute embeddings.

Using the attribute embeddings, calculate two matrices of vector representations for entities KG_1 and KG_2, denoted by $E^{(1)}_{AE} \in \mathbf{R}^{n^{(1)}_e \times d}$ and $E^{(2)}_{AE} \in \mathbf{R}^{n^{(2)}_e \times d}$ respectively. In the matrices, rows are represented be vector representation of entities, which is calculated as $e = [\sum_{a\in A_e} a]_1$ where A_e is the set of attributes of e and $[.]_1$ denotes the normalized vector. Using these, cross-KG similarity matrix $S^{(1,2)} \in \mathbf{R}^{n^{(1)}_e \times n^{(2)}_e}$, and inner similarity matrices $S^{(1)} \in \mathbf{R}^{n^{(1)}_e \times n^{(1)}_e}$, $S^{(2)} \in \mathbf{R}^{n^{(2)}_e \times n^{(2)}_e}$ will be calculated. These matrices are defined as follows;

$$S^{(1,2)} = E^{(1)}_{AE} E^{(2)\,T}_{AE}, \;\; S^{(1)} = E^{(1)}_{AE} E^{(1)\,T}_{AE}, \;\; S^{(2)} = E^{(2)}_{AE} E^{(2)\,T}_{AE} \tag{3.8}$$

The similarity matrix S holds the cosine similarities among entities and $S_{i,j}$ is the similarity between the i-th entity in one KG and the j-th entity in the same or the other KG. Note that cosine similarity of two entities e, e', is calculated by $cos(\mathbf{e}, \mathbf{e}') = \frac{\mathbf{e}.\mathbf{e}'}{\|\mathbf{e}\|.\|\mathbf{e}'\|} = \mathbf{e}.\mathbf{e}'$ as entity vectors are normalized (lenght of of any embedding vector is enforced to 1).

Now, the aim is to cluster similar entities across KGs together so that their embeddings would be similar. For that following loss function is minimized.

$$L_S = \|E^{(1)}_{SE} - S^{(1,2)} E^{(2)}_{SE}\|^2_F + \beta\left(\|E^{(1)}_{SE} - S^{(1)} E^{(1)}_{SE}\|^2_F + \|E^{(2)}_{SE} - S^{(2)} E^{(2)}_{SE}\|^2_F\right) \tag{3.9}$$

Where β is a hyper-parameter that balances similarities between KGs and their inner similarities. $E_{SE} \in \mathbf{R}^{n_e \times d}$ denotes the matrix of entity vectors for one KG in SE with each row an entity vector. In the loss function (Eq 3.9), alignment term is $\|E^{(1)}_{SE} - S^{(1,2)} E^{(2)}_{SE}\|^2_F$, which by minimizing it, similar entities across KGs will be embedded similarly.

Finally, to preserve both the structure and attribute information of two KGs, jointly minimize the following objective function;

$$L_{joint} = L_{SE} + \delta L_S \tag{3.10}$$

where δ is a hyper-parameter weighting L_S.

3.2.4 GCNAlign

GCNAlign [8] model is based on the Graph Convolution Networks (GCN). Therefore, before moving into GCNAlign, let's look at the theory behind GCNs [20].

GCN is a neural network type that was designed to implement on graph data. GCN takes feature vectors of nodes and the structure of the graph as input. Then it will learn a function of features on input graph and output embeddings of nodes. These embeddings of the nodes will capture the information about the neighborhood of a node (structure information). Also, GCN combines this structure information with attribute information of a node. In KG alignment, it is assumed that equivalent entities have similar attributes and similar neighborhoods. So the GCN is idle for KG alignment as it captures the attributes and similarity of the nodes.

GCN contains multiple stacked GCN layers. Layer-wise propagation rule of GCN is defined as below;

$$H^{(l+1)} = \sigma\big(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\big) \tag{3.11}$$

Here $H^{(l)} \in \mathbf{R}^{n\times d^{(l)}}$ is input to the l-th layer, where n is the number of nodes and $d^{(l)}$ is the number of features in l-th layer. Further,$\sigma(.)$ is an activation function chosen as $ReLU = max(0,.)$; A is a $n\times n$ connectivity matrix(more details below) that represent the structural information of the graph; $\hat{A} = A + I$, and I is the identity matrix; $\hat{D} =$ is the diagonal node matrix of $\hat{A}$, where diagonal element is defined as $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$; $W_{(l)} \in \mathbf{R}^{d^{(l)}\times d^{(l+1)}}$ is the weight matrix of the l-th layer, $d^{(l+1)}$ is the dimensionality of new node features.

To include both structures and attribute information in GCN, two feature vector for each entity is used, denoted as h_s and h_a for structure and attribute feature vectors respectively. In the input layer,$h_s^{(0)}$ is randomly initialized and updated in training whereas $h_a^{(0)}$ is the attribute vectors of entities and fixed during the training. Then let H_s and H_a be structure and attribute feature matrix, which consists of feature vectors of entities (each row corresponds to feature vector of an entity). Now layer-wise propagation is modified as;

$$[H_s^{(l+1)}; H_a^{(l+1)}] = \sigma\Big(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}[H_s^{(l)}W_s^{(l)}; H_a^{(l)}W_a^{(l)}]\Big) \tag{3.12}$$

Where $W_s^{(l)}$ and $W_a^{(l)}$ are weight matrix for structure and attribute features in the l-th layer respectively; [;] denotes concatenation of two matrices.

GCNAlign Implementation

In GCNAlign, GCN is implemented on each KG separately to generate embeddings of entities. The number of layers used in GCN is two. Initially, input the structure matrices and attribute matrices for each KG with the connectivity matrix. The dimensionality of structure feature vectors is set to d_s and kept same on all the layers for both GCNs, and structure weight matrices are shared among two GCNs for two layers, $W_s^{(1)}$ and $W_s^{(2)}$. For attribute feature vectors, input attribute vector dimensionality can be different because two KGs may have a different number of attributes. So the first layer of each GCN transforms input attribute feature vector to dimensionality d_a, so that two GCN models give attribute embedding of the same dimensionality. Final output is entity embeddings of $d_s + d_a$ dimensionality.

As per **Connectivity matrix** A, it is computed as follows. Let $a_{ij} \in A$, denotes the amount of information propagates from i-th entity to the j-th entity. To compute this, two measures for each relation called functionality and inverse functionality is computed.

$$fun(r) = \frac{\#Head\_Entities\_of\_r}{\#Triples\_of\_r} \tag{3.13}$$

$$ifun(r) = \frac{\#Tail\_Entities\_of\_r}{\#Triples\_of\_r} \tag{3.14}$$

where $\#Triples\_of\_r$ is the number of triples of realtion r; $\#Head\_Entities\_of\_r$ and $\#Tail\_Entities\_of\_r$ are the no of head entities and tail entities of r respectively. Then a_{ij} is calculated as follows;

$$a_{ij} = \sum_{(e_i,r,e_J)\in G} ifun(r) + \sum_{(e_j,r,e_i)\in G} fun(r) \tag{3.15}$$

Connectivity matrix will inlude the a_{ij} as elements. Use of this method allows to capture the structure in a KG which is relational multi-graph where entities are connected by relations.

Now the GCN models are trained by minimimzing below margin-based ranking loss functions with the set of known alignment S as training data.

$$L_s = \sum_{(e,v)\in S} \sum_{(e',v')\in S'_{(e,v)}} [f(\boldsymbol{h}_s(e), \boldsymbol{h}_s(v)) + \gamma_s - f(\boldsymbol{h}_s(e'), \boldsymbol{h}_s(v'))]_+ \tag{3.16}$$

$$L_a = \sum_{(e,v)\in S} \sum_{(e',v')\in S'_{(e,v)}} [f(\boldsymbol{h}_a(e), \boldsymbol{h}_a(v)) + \gamma_a - f(\boldsymbol{h}_a(e'), \boldsymbol{h}_a(v'))]_+ \tag{3.17}$$

where $[x]_+ = max(0, x)$, $S'_{(e,v)}$ denotes set of negative (corrupt) alignments constructed by randomly replacing entities of alignment set, and $\gamma_s, \gamma_a > 0$ are margin hyper-parameters. L_s and L_a are loss functions for structure and attribute embeddings, which are independent and hence optimized separately.

We will implement GCNAlign model as implemented in the paper [21]. We do not use attribute features. So the implementation is much simpler and follows the same steps described above without considering the attribute features. Connectivity matrix is computed with setting $fun(r)$ and $ifun(r)$ at 0.3 as minimum. Further input node features are normalized to unit euclidean length. Finally, there are no convolution weights $W_s^{(1)}$ and $W_s^{(2)}$. This indicates that GCN is just a fixed function on the learned node embeddings. The final output from GCNAlign model applied to two KGs with prior alignment set is embedding of entities (we set embedding dimension to 200) of both KGs in a unified vector space. For aligned entities, it is expected that these embeddings are pretty similar.

3.2.5 Comparison Between Alignment Models

When compare among the first three approaches JE, MtransE, and JAPE, they all have a somewhat similar implementation. They all use a version of TransE to learn embeddings of entities and define some transformation between the embeddings of aligned entities. As all these above methods use TransE to embed entities initially, all these approaches also embed relations. Also, JE and JAPE use negative triples but MTrasE does not, which is reflected in the respective loss functions used. As per the use of attribute information, JAPE facilitates the use of it, whereas JE and MTransE do not. Further, all these approaches need some initial alignment data in the form of triples, between KGs to successfully implement the model to identify alignments.

When compare the first three approaches with the GCNAlign (model implemented in the study), there are certain differences. GCNAlign uses graph convolution networks (GCNs) for embedding instead of using TransE. Therefore GCNAlign does not learn embeddings of relations and only embed the entities. Further, GCNAlign also supports using attribute features in the model similar to JAPE. As per initial alignment data, GCNAlign only needs a list of align entities of two KGs, it does not require aligned triples as other models. Therefore, when consider all these comparisons among different approaches, GCNAlign seems to be the most flexible and simple method. Also, the accuracy of GCNAlign is high compared to other methods discussed. Hence the decision to use GCNAlign in our study.

3.3 Active learning for Knowledge Graph Embedding

As discussed earlier, active learning can be a solution to the main problem associated with the above alignment models, lack of initial alignment data. As per my knowledge, there exists no study that

directly investigates the use of active learning for entity alignment task. However, there are some studies that explore the use of Active Learning with respect to KGs.

For example, paper [22] discuss using active learning for graph embedding. Authors implement multiple active learning query strategies. Query strategies are based on two main categories; graph structure and learned node embeddings. Under graph structure, centrality measures and under node embeddings, node classification uncertainty and node density are used to rank the nodes. In centrality measures AL framework, every nodes' centrality is calculated using page rank and the nodes with the highest centrality value are chosen first. In the uncertainty AL framework, the uncertainty of each node is calculated using entropy and the node with the highest entropy (or highest uncertainty) is chosen. In the density AL framework, each node density is calculated and nodes with the highest density are chosen. Furthermore, paper introduces an AL framework combining all these strategies named "Active Graph Embedding (AGE)". AGE uses all three strategies to rank the nodes, and the weightings of each strategy depend on the number of queries. AL framework is defined as follows; high weight for centrality measure and low weights for uncertainty and density when the number of queries is low. When number of queries is high, density and uncertainty measures get high weight and centrality measure gets low weight. The reason for this is that uncertainty and density are calculated using node embeddings, which is updated with each query, whereas centrality does not use embeddings. These embeddings at initial stages will not be very accurate because of limited training data, however as AL framework query more instances, more training data, and more accurate embeddings. This is the main idea of the AGE framework. The implementation of these AL frameworks was done using GCNs, which closely parallels the GCNAlign (the model used in the study for KG alignment). Therefore, these ideas and concepts in the paper were very influential in the design of AL frameworks used in our study.

Chapter 4

Settings and Implementation

This chapter gives details of the settings for the experiments conducted and the implementation of the experiments. This includes a detail description of the implementation of query strategies, oracle, model and performance evaluation. Further, descriptions about the data-sets used for the experiments also will be given. Therefore the chapter can be considered as a link that will combine all the preceding and subsequent chapters.

4.1 Experimental Setup

The framework on how the experiments were conducted, is given in Figure 4.1; In the figure, all the elements inside the grey box can be considered as the major components in the framework. The first step of the setup is to obtain the KG Alignment data-set. Once the data-set is obtained, we will organize the data-set such that we have two KGs and Alignment data. Alignment data is further divided to train, validation and test data. The next step is the AL framework, which consists of two components; query system and oracle. Data-set will be provided to both the query system and oracle. First, the query system will query the nodes according to an AL heuristic. Then, the query system will send the selected nodes to the oracle to label (label means finding selected nodes' all matching nodes in the other KG). Here we simulate the oracle to identify the label of a node by feeding the data-set to oracle. Also, the oracle will provide the query system with specific nodes to consider for querying (referred to as available nodes). This is to avoid querying the nodes that were already queried. After the oracle label the nodes, align entities will be used to fit the model. Here we use the validation data to use in early stopping. After the model fit, embeddings will be used to evaluate performance with test data. Note that, some AL heuristic use embedding for querying of the nodes. Arrows in the figure which are drawn with dotted lines indicate that the process represented will not always happen. All these steps will be discussed in much more in-depth in the implementation section.

4.2 Datasets

Before moving to the implementation, we will discuss the two data-sets used in the study DBP15K and WK3l-15K.

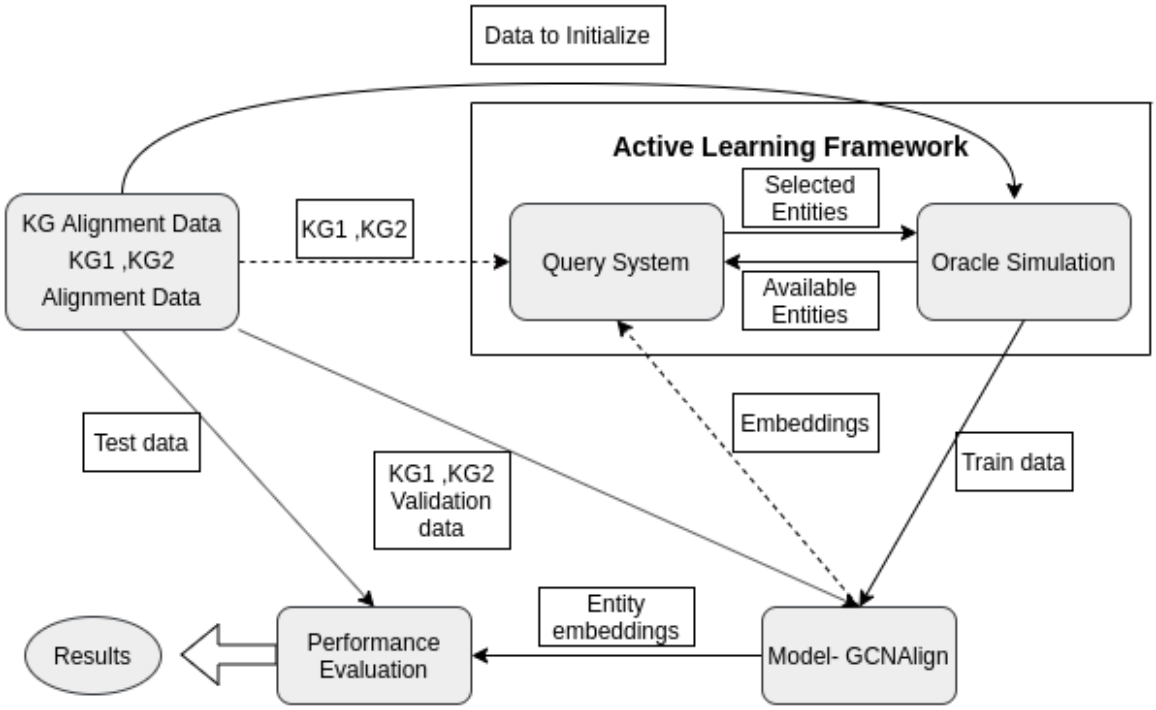


Figure 4.1: Implementation framework in the study

4.2.1 DBP15K

DBP15K data-sets were built for JAPE paper [7]. The data-sets were created using DBpedia, which is a large-scale multi-lingual KG. It includes Inter-Language Links (ILLs) between different languages. To build the data-sets, 15 thousand ILLs pairs with popular entities (entities with at least 4 relationship triples) from English to Chinese, English to Japanese and English to French were extracted. Table 4.1 gives the number of elements in each set of the specific KG (as defined in section 2.1).

Table 4.1: Details of the data-set DBP15K

Dataset		Entities	Relations	Attributes	Rel. triples	Attr. triples
DBP15K	Chinese	66,469	2,830	8,113	153,929	379,684
	English	98,125	2,317	7,173	237,674	567,755
DBP15K	Japanese	65,744	2,043	5,882	164,373	354,619
	English	95,680	2,096	6,066	233,319	497,230
DBP15K	French	66,858	1,379	4,547	192,191	528,665
	English	105,889	2,209	6,422	278,590	576,543

On top of this data-set, JAPE authors have also built another 5 data-sets to use for the JAPE model. We will use one of these subsets in our study. Table 4.2 gives details of the data-sets we use in DBP15K. Since we don't use any attribute information in the model, we ignore any information

about it.

Table 4.2: Details of DBP15K subsets used in the study

Dataset		Entities	Relations	Triples	Aligned test	Aligned train	Aligned validation
DBP15K	Chinese	19,388	1,701	70,414	10,500	3,600	900
	English	19,572	1,323	95,142			
DBP15K	Japanese	19,814	1,299	77,214	10,500	3,600	900
	English	19,780	1,153	93,484			
DBP15K	French	19,661	903	105,998	10,500	3,600	900
	English	19,993	1,208	115,722			

As shown in Figure 4.1, we will organize the data-set as two KGs and Alignment data-set. Two KGs can be referred to as Match KG (or left KG) and Reference KG (or right KG). The alignment model will try to match the entities of match KG with the reference KG and vice versa. For alignment data, data-sets consist of 15,000 alignment entity pairs for each language pair, which is divided into training, testing and validation as shown in Table 4.2. This data-set is available in this website[1].

4.2.2 WK3l-15K

WK3l data-set was built for the MTransE paper [6]. It contains English (En), French (Fr), and German (De) knowledge graphs under DBpedia's dbo: Person domain. To construct the alignment data-set, triples are aligned by verifying the ILLs on entities, and multilingual labels of the DBpedia ontology on some relations. Using this, two data subsets, WK3l-15K and WK3l-120K are constructed. Statistics of these data-sets is given below.

Table 4.3: Details of the data-set WK3l

Data-set	#En triples	#Fr triples	#De triples	#Aligned triples
WK3l-15K	203,502	170,605	145,616	En-Fr: 16,470 En-De: 37,170
WK3l-120K	1,376,011	767,750	391,108	En-Fr: 124,433 En-De: 69,413

For our study, we use WK3l-15K data-set and it is organize according to our requirements. Details of the organized data-set is given below;

Table 4.4: Details of WK3l-15K subsets used in the study

Data-set		Entities	Relations	Triples	Aligned test	Aligned train	Aligned validation
WK3l-15K	English	15,127	1,841	209,041	7,268	2,492	623
	German	14,603	596	144,244			
WK3l-15K	English	15,170	2,228	203,356	5,617	1,925	482
	French	15,393	2,422	169,329			

[1] `https://github.com/nju-websoft/JAPE`

4.3 Implementation

Below, the implementation of the components of our framework is discussed. Specifically, We will give details on the following; how querying of instances is done using AL, how the oracle operates, how the model is implemented and how the performance is evaluated.

4.3.1 AL Framework Implementation

Once the data is obtained and organized, it is feed to the AL framework (as shown in Figure 4.1). AL framework consists of two components; 1) query system, 2) oracle. Data is feed to both query system (not always) and oracle separately. To initialize the oracle, full KG Alignment data-set is fed into the oracle. Query system will get the nodes to consider for querying from the oracle. Some AL heuristics (centrality-based heuristics) require the structure of the KGs, in that case, that is provided by feeding it the two KGs of the data-set. Further, some AL heuristic requires embeddings of the entities in the implementation. In that situation, it is provided by the fitted model. These are shown with the dotted arrows in Figure 4.1.

4.3.1.1 Query System

The query system is the main component of the AL framework. First, the query system takes all candidate nodes to label from the oracle. Query system includes multiple AL heuristics, which will choose a node to label, where labeling is done by the oracle. This choosing of the node is done by ranking all the available nodes. This is the pool-based scenario in the AL setting. We have implemented multiple methods to do this ranking of the nodes. Those methods are discussed below.

1) Centrality-Based Active Learning

This is the most simple implementation of AL heuristic in the study. We will calculate the centrality of each node using a centrality measure. Then the node with the highest centrality is chosen as the node to label. We use five different centrality measures; Degree, Betweenness, Page Rank, Closeness and Harmonic. All these measures are defined in Chapter 2. Calculation of centrality measures was done using the python package networkx[2].

If we consider the query strategies discussed in section 2.3.3, this can be considered as an AL implementation that uses the structure of the data. .

2) Uncertainty Sampling

Uncertainty Sampling is the most common implementation of AL. As discussed in section 2.3.3, we need a posterior distribution to calculate the uncertainty of an instance. For this, we will define a posterior distribution for each candidate nodes (i.e nodes that will be queried by the system) in both KGs. To derive the posterior distribution of a node, we calculate the similarities between that node and all other candidate nodes in the other KG given node embeddings. Node embeddings are derived from the fitted GCNAlign model. Figure 4.1 shows this use of embeddings with the dotted arrow from GCNAlign to Query System. Assume that the probability that two nodes from different KGs are aligned is the similarity between them. However, these similarities calculated for a specific node cannot be considered as its' posterior distribution yet, because these similarity values are not normalized (does not add up to one). We will derive two uncertainty sampling implementations with the way we normalize the similarity values. Once we have the posterior probabilities, we can

[2] `https://networkx.github.io`

calculate an entropy score associated with posterior probability distribution of each node. Then the higher entropy score, higher priority in choosing in AL framework.

We implement two uncertainty sampling methods by normalizing the similarity values of each node and using it to calculate the entropy of the node. Entropy equation is given in the equation 2.3.

- For each node, get the maximum of the similarities calculated with nodes from other KG and use it to calculate the entropy score. To calculate the entropy score we need to normalize the probabilities associated with the node. Therefore we define the posterior distribution of each node such as; (maximum similarity, 1-maximum similarity).
- For each node, get the maximum k number of similarities calculated with nodes from other KG. Use these k similarities to calculate the entropy score. To normalize the similarity values, we apply softmax function before calculating entropy. Also, we will set k=10.

In uncertainty sampling, we are not looking for the nodes that have the highest probability to have an alignment in other KG. We want to query the nodes which model is most uncertain about having an alignment. In theory, these nodes should have the highest information about the alignments between two KGs. We will investigate whether this holds up for our situation also in the section 5.2.

3) Information Density

Density-based heuristic is built on the idea of exploiting the structure of data. As discussed in Chapter 2, we want to incorporate this idea into the query the nodes from the dense region in KGs. As given in equation 2.8, we will use generic information density heuristic implementation. For the information part (given by $\phi_A(x)$) in the equation, we will use a version of uncertainty sampling. The information density of a node is calculated as follows;

1. For a node in a KG, calculate the similarities with all other nodes in the same KG and get the average similarity. This is the density part of the equation.
2. For a node in a KG, calculate the average of the similarities with nodes from other KG. Use it to calculate the entropy score (same as maximum similarity method in uncertainty sampling implementation). This is the information part of the equation.
3. We will set $\beta = 1$ in the equation 2.8. This indicates that equal importance is given to both information and density parts in querying. Therefore to get the information density of a node, we will simply multiply information and density scores.

Then, the higher the information density of a node, the higher the priority in querying.

4) Cluster-Based Active Learning

Cluster-based AL can be used to retrieve information of the structure of the data. To implement this, we use the embeddings derived from the model. So the algorithm to get cluster score of each node is given below;

1. Get the embeddings of available nodes of two KGs separately.
2. Apply K-means clustering on those two sets of embeddings separately and derive clusters and cluster centroids.

3. For each node, calculate the similarity between the node and its' corresponding cluster centroid.

We take the similarity of the node with its' cluster centroid as the node cluster score. A higher cluster score means that the node is more closer to its' cluster centroid. Then the nodes with high cluster scores are given priority when querying. To apply K-means we set the number of the cluster as the square-root of the number of available nodes [23].

5) Maximum Shortest Path Distance

Maximum Shortest Path Distance (MSPD), try to infer information from the nodes that are found to be aligned. For the implementation we use the multi_source_dijkstra[3] algorithm in networkx. This function will return the shortest paths and lengths between one of the source nodes and all other reachable nodes. Here, as source nodes, we provide the aligned nodes and get the shortest path distance for all available nodes with one of these aligned nodes. For example, consider a node, the function will compute the shortest path distance of this node with all the aligned nodes (source nodes) and return the smallest of these distance for that node. If there is no shortest path between the node and all source nodes, it will not be considered for querying. After getting the shortest path distance for all the available nodes, we will select the nodes that have higher values for the distance first.

We will discuss more about implementation steps of AL strategies and reasons for taking these steps in more detail in section 6.2.

4.3.1.2 Oracle Simulation

To simulate the task of the oracle in AL Framework, we implemented an Oracle that does the following main tasks;

1. Provide the query system with the candidate nodes to select
2. Label the queries that are selected by the query system
3. provides the training data to GCNAlign-model.

When the oracle receives the data-set initially, it will allocate the nodes in the testing and validation to the forbidden set that cannot be accessed by the query system, and all other nodes will be available to query system to query. Once the query system sends a node to label, oracle will look at the training alignment data it has and gives labels to that node. That is whether there are align nodes in the other KG for the node selected by the query system. So if there is an alignment, then those align pairs will be put into a set(referred as alignment set) and if there is no alignment, that node put to another set (referred as exclusive set). Also, in both those situations, these nodes will be removed from the available set. We do not use exclusive set in our implementation. But we give future implementation suggestions on a method to use it in section 6.6. Finally, the oracle provides alignment data (node pairs in alignment set) as training data for the GCN-Align model to fit the model.

[3] `https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.weighted.multi_source_dijkstra.html`

4.3.2 GCNAlign Implementation

Once the oracle label some set of nodes and send the alignment set, we can use the alignment set to fit the model. Oracle labeling of the nodes is done in step sizes, which will be explained in the next section, Performance Evaluation. Using this alignment set, loss function defined in equation 3.16 will be minimized. Apart from the alignment set, we also need a negative sample set for the loss function. To get the negative sample, for each aligned pair in alignment set, we randomly replace one of the nodes with another node from the same KG. To make this simpler for explanation, consider keeping all the left KG nodes in align set same and replace the right KG nodes with some other nodes from right KG (for training we use both sides). When replacing, we avoid nodes that are in exclusive set. For the minimization of the loss function, we will use the optimization algorithm Adam [24]. Also, we will use the regularization technique of Early stopping. To implement early stopping we will use the validation data set. For other parts in the GCNAlign model (fitting GCNs for KGs) which is explained in section 3.2.4, we will use the KG-Alignment data-set, as shown in Figure 4.1.

4.3.3 Performance Evaluation

One of the main requirements of the study is to study the use of AL heuristic to entity alignment. For this, we plot the performance with number of AL queries. We will define a set of step sizes and evaluate the performance after each step size. Step sizes are the total number of AL queries considered at a specific point. The difference between the step-sizes will be small at the beginning and become larger as the value of step size increases. For example, step size list will look like this; (20,40,60,80,100,150,200,...). The final step size would be something closer to the number of all available nodes for querying. However, we stop the process without going through all step sizes if we locate all the align data before ruining that many queries. Our implementation will be as follows; first, ask for 20 AL queries, and label those queries, fit the model with these queries and calculate the performance and store the results. Next step size is 40, therefore we ask for another 20 queries (the difference between step size) and repeat the same steps described. Do this for all the step sizes in the list. For performance evaluation we will use the metrics defined in section 2.4.3.

Chapter 5

Analysis and Results

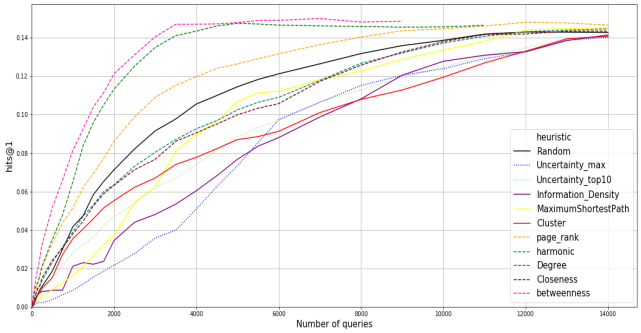
This chapter gives details on the analysis of the data and the result derived. As per the objectives, we first compare the performance of active learning heuristics implemented for five different data-sets. We will use several methods to do this, one is, we plot the performance of each AL heuristic with the number of queries. Also, we plot the performance with the percentage of the training data used. We will use several performance measures to make sure that performance is consistent across all the measures. Also, the use of multiple data-sets will validate the consistency of the results obtained.

5.1 Performance of AL Heuristics with Number of Queries

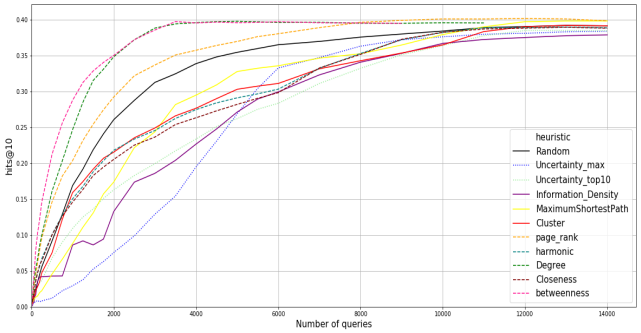
To identify the performance of AL heuristics, we analyze the performance of each heuristic with respect to the number of queries. Four performance measures used are Hits@1, Hits@10, Hits@50 and mean reciprocal rank (MRR). To act as a base to compare performance, we used a random heuristic. In random heuristic, instead of using AL heuristic to select an instance (node) to label, an instance will be randomly selected. Performance evaluation will be similar to that of using an AL heuristic. AL heuristic to be useful, its' performance should be better than that of random heuristic.

We will first look at the performance of AL heuristic with DBP15K data-sets. As shown in the figures 5.1, 5.2 and 5.3, we can see that best performing AL heuristics are degree, betweenness and page rank for all data-sets in DBP15k over all performance measures. All other heuristics considered do not consistently outperform Random heuristic. When consider among the better-performing heuristics, degree and betweenness significantly perform better than the page rank. Among degree and betweenness, betweenness slightly performs better than degree. These results are very consistent across all measurement types and DBP15K data-sets.

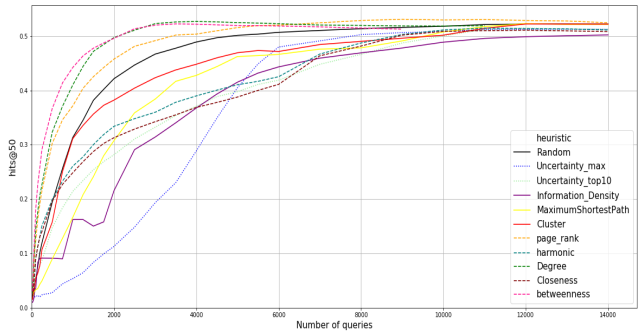
When consider among heuristics that do not out-perform random, all embedding based heuristics(two uncertainty sampling, cluster and information density implementations) perform noticeably worse at the start than other heuristics. This is expected as, at the beginning loss function is minimized with limited training data, therefore the embeddings derived from the model will be not very accurate. However, our expectation was that once some training data was used, performance of these heuristics should increase significantly and performs better or at least on-par with the other heuristics. However, this is not the case as it takes a significant number of queries (around 6000-8000) for these heuristics to come on-par with the other heuristics.



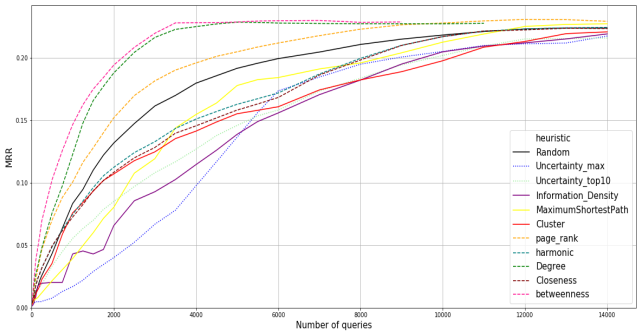
(a) Performance measure- hits@1



(b) Performance measure- hits@10

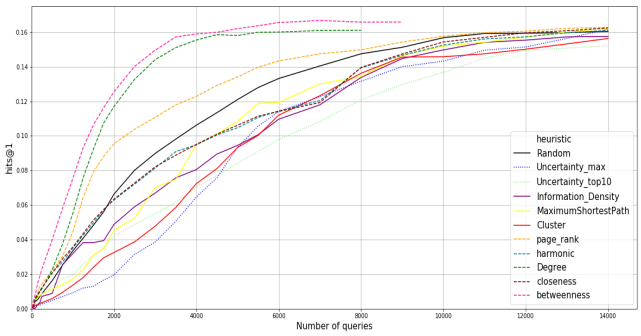


(c) Performance measure- hits@50

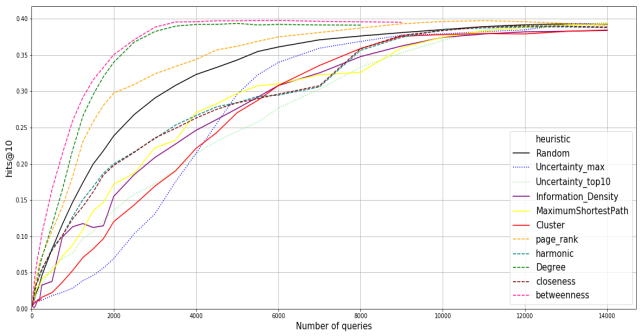


(d) Performance measure- MRR

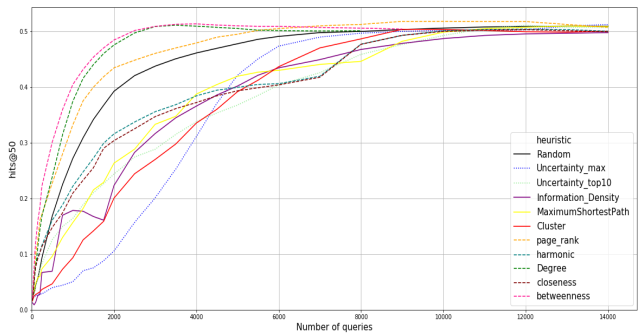
Figure 5.1: Performance of AL heuristics for dataset DBP15K- French to English



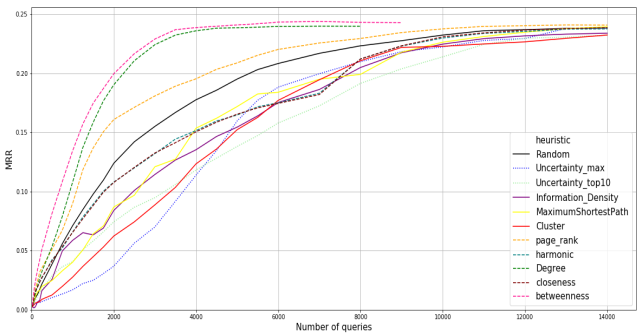
(a) Performance measure- hits@1



(b) Performance measure- hits@10

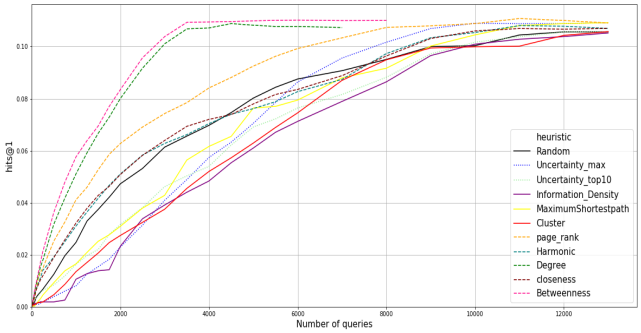


(c) Performance measure- hits@50

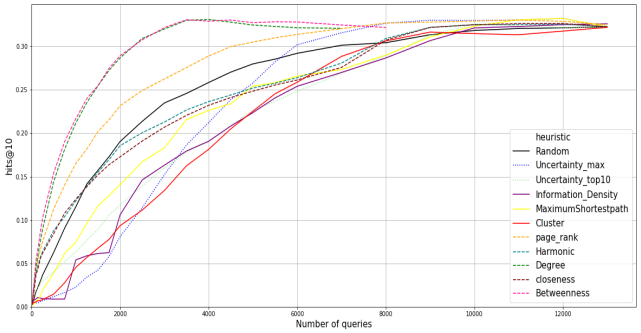


(d) Performance measure- MRR

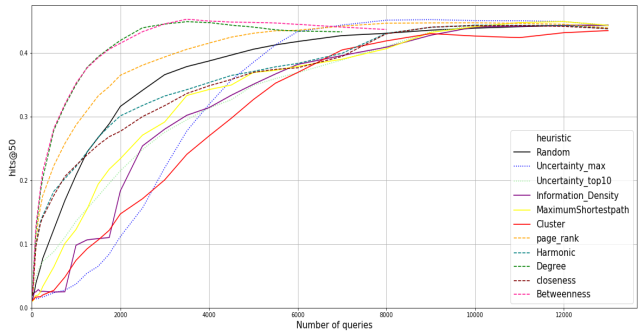
Figure 5.2: Performance of AL heuristics for dataset DBP15K- Japanese to English



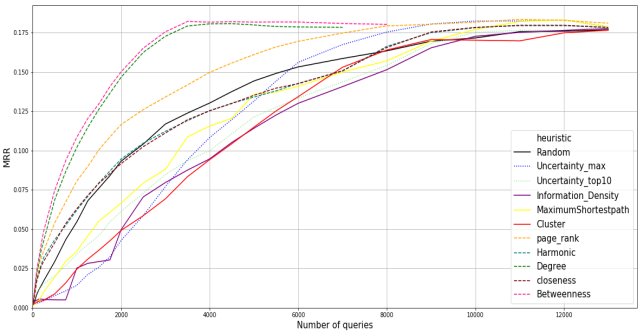
(a) Performance measure- hits@1



(b) Performance measure- hits@10

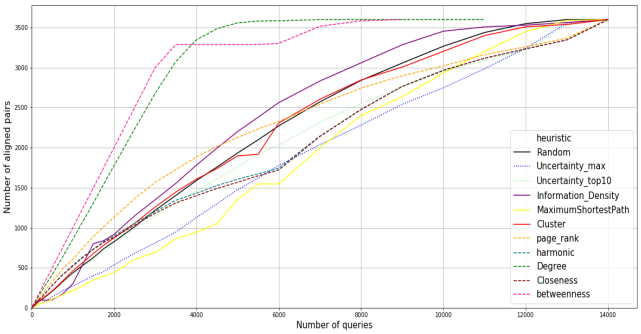


(c) Performance measure- hits@50

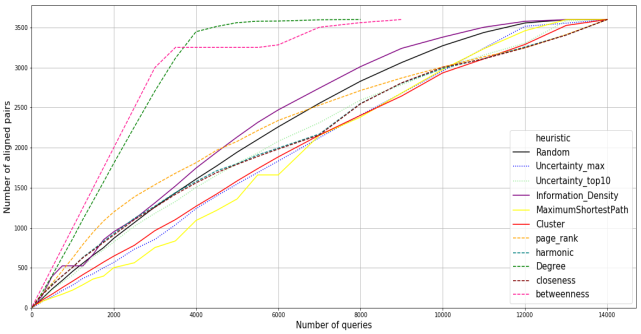


(d) Performance measure- MRR

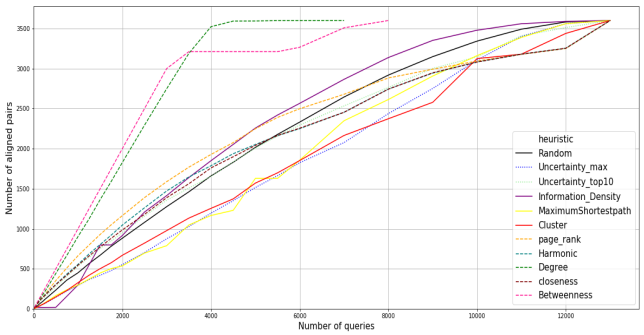
Figure 5.3: Performance of AL heuristics for dataset DBP15K- Chinese to English



(a) Data-set DBP15K-French to English



(b) Data-set DBP15K-Japanese to English



(c) Data-set DBP15K-Chinese to English

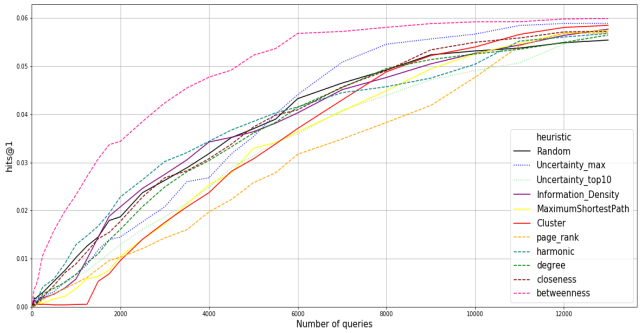
Figure 5.4: Number of alignment pairs with number of queries-DBP15K

Further, note that in degree and betweenness, all the aligned data is found before other heuristics. That is the reason that the total number of queries for these heuristics are less than other heuristics. This means that these heuristics queries' are more successful in finding alignment pairs than other heuristics. The reason for this high success will be discussed in section 5.4. Figure 5.4 plots the number of alignment pairs with the number of queries, which clearly shows that degree and betweenness have a significantly higher success rate than other heuristics. This is the main reason for the better performance of degree and betweenness over other heuristics. When consider the page rank, which has a better performance than random, we can see a slightly higher success rate initially over other heuristics except for degree and betweenness, which can be attributed to this better performance of page rank than random. For other AL heuristics, the success rate of finding aligned pairs is pretty much similar to each other. For most of these heuristics success rate at the start (until around 2000 queries) is below or pretty similar to that of random. This may be the reason for them to not perform better than random. To validate the results we derived so far, we will get the performance of AL heuristics with another data-set, WK3l. We will do the same steps did above and derive similar plots for it also.

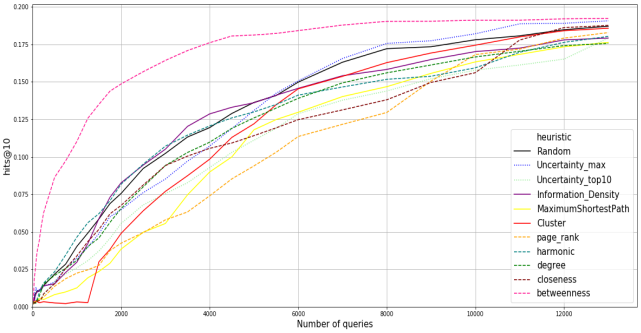
Figures 5.5 and 5.6 depicts the performance evaluation of each AL heuristics derived for two data-sets of WK3l. Results derived for these data-sets are noticeably different from that of DBP15K. For the data-set Wk3l-English to German (Figure 5.5), only AL heuristic that seems to have a significantly better performance than random heuristic is the betweenness. All other heuristics do not perform better than random consistently over all the measurement types considered. However, for the data-set Wk3l-English to French (Figure 5.6), results seem a little different. It seems that betweenness, degree, page rank and maximum shortest path distance heuristics have better performance than random for all measurement types. However, betweenness clearly out-performs other heuristics significantly.

To identify the rate of query success we plot the number of queries with the number of aligned pairs for the two data-sets of WK3l, which is given in Figure 5.7. The query success rate of betweenness is slightly better than that of random in the data-set WK3l-English to German. However there is other AL heuristics that have a better query success rate than random for that data-set, but any heuristic other than betweenness does not significantly outperform random heuristic in terms of performance. And when consider the query success-rate for data-set WK3l-English to French, betweenness clearly out-performs other heuristics including random at the beginning to until around 5000 queries. Other heuristics such as degree and page rank also have a significantly higher query success rate than random at the beginning. This may explain the better performance of page rank and degree over random for this data-set. When consider the other noticeably better-performing heuristic for this data-set, which is the maximum shortest path distance, its' query success rate is very similar to random. However, the maximum shortest path distances' only perform better than random only for the data-set WK3l-English to French. Therefore we can conclude that its' performance is not consistent.

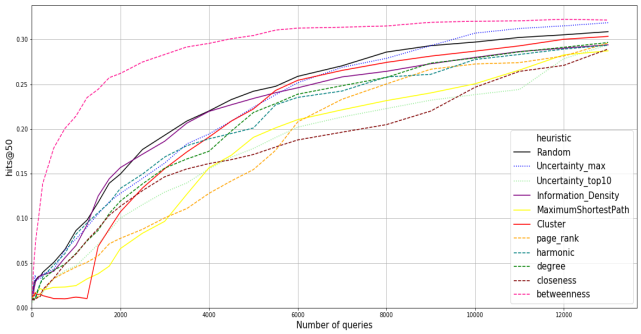
So when we consider the performance of each AL heuristic over all five data-sets, the only heuristic that consistently and significantly outperform random heuristic is betweenness. Heuristics degree and page rank out-perform random in four data-sets out of five. All the other heuristics do not consistently and significantly outperform random heuristic. We identified that one of the main factors of the performance of AL heuristic is the query success rate. However, we found that even-though query success rate is higher than random heuristic, it does not always guarantee better performance than random heuristic (Figure 5.5). Therefore we can derive that the performance of AL heuristic will depend on two factors; Rate of successful queries and information contained on successful queries. Therefore, AL heuristic to have a good performance, it has to choose instances that are not only more likely to have an alignment but also contain high information content.



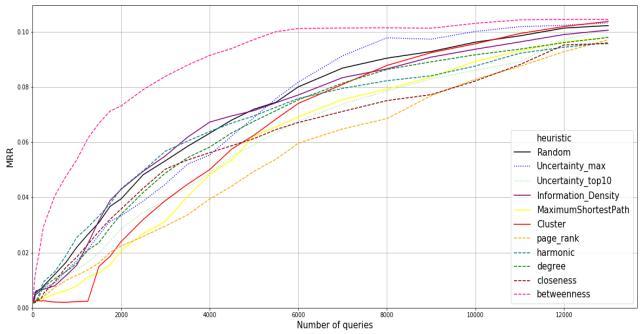
(a) Performance measure- hits@1



(b) Performance measure- hits@10

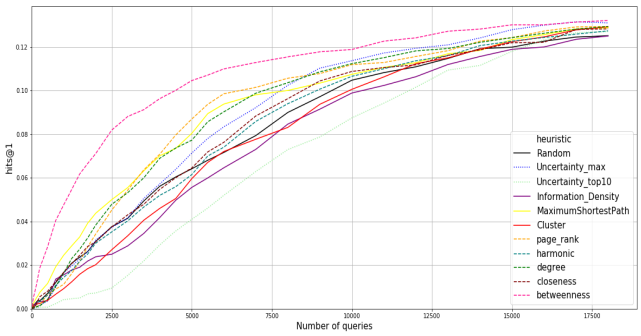


(c) Performance measure- hits@50

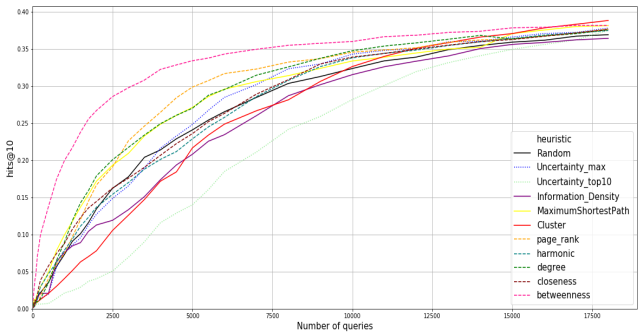


(d) Performance measure- MRR

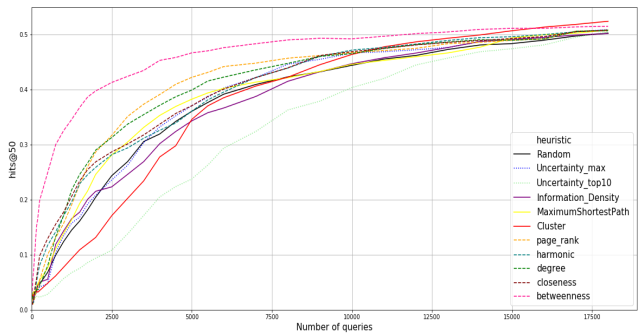
Figure 5.5: Performance of AL heuristics for dataset WK3l- English to German



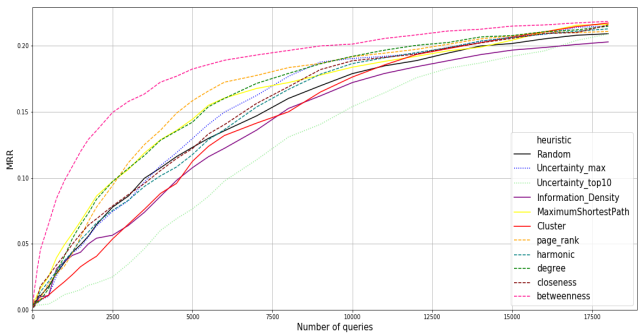
(a) Performance measure- hits@1



(b) Performance measure- hits@10

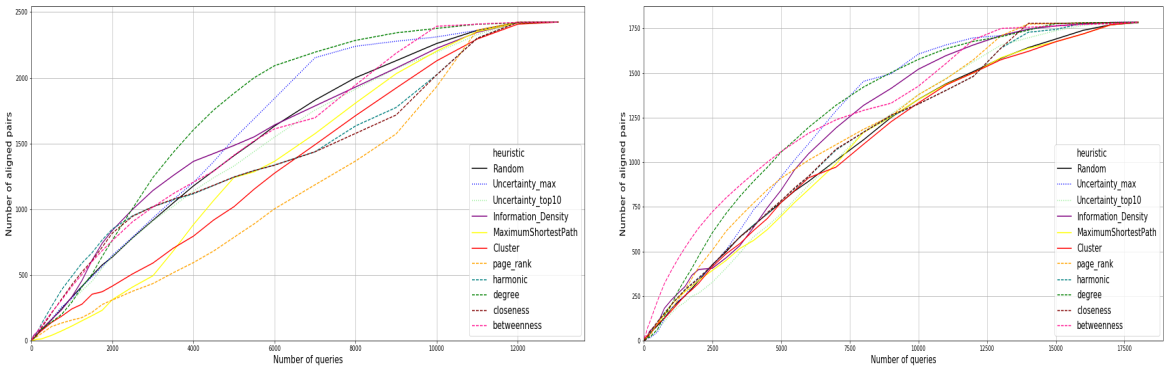


(c) Performance measure- hits@50



(d) Performance measure- MRR

Figure 5.6: Performance of AL heuristics for dataset WK3l- English to French



(a) Data-set WK3l- English to German

(b) Data-set WK3l- English to French

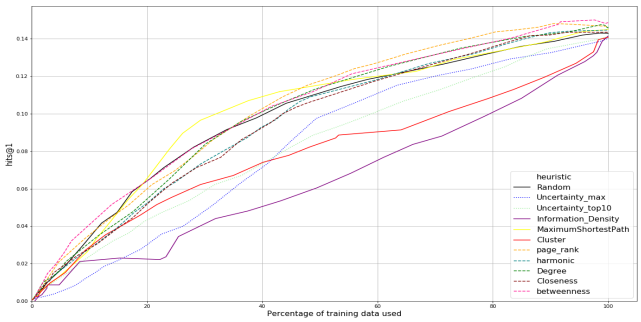
Figure 5.7: Number of alignment pairs with number of queries- WK3l

5.2 Performance of AL Heuristics with the Training Data Size

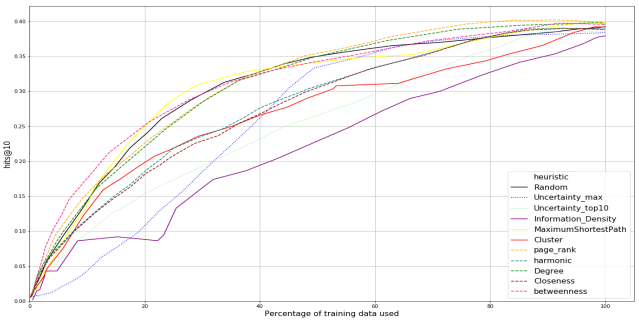
In our AL implementation, if a query returns a node that does not have an alignment, we will ignore it. Hence, it is important that AL heuristics query the instances that are more likely to have an alignment. However, most existing AL heuristics in literature aims to query the instance with the most information. In the above section, we also identify that the information contained in the query instance has a role in the performance of AL heuristic. To study this effect, we will plot the performance with the percentage of training data used. As similar to number of queries, AL heuristic should perform better than random heuristic with respect to data percentage used for considered to be a useful heuristic. Further, if AL heuristic chooses nodes with more information, then it should have a higher rate of performance increase at the start. That is, the slope of the plot of performance with the percentage of training data used should be decreasing.

We will first, examine performance with percentage of data used for DBP15K data-set. Figures 5.8, 5.9 and 5.10 plots the performance with data percentage used for the three sub-sets of data in DBP15k. If we carefully examine all these plots, we can see that betweenness heuristic performance is consistently better than other heuristics at the beginning (up until around 20% of data is used). This better performance is not very significant, nevertheless, it is clear that betweenness performs better than random with respect to data percentage used. Other heuristics do not seem to out-perform random consistently. As similar to number of queries and performance, embedding-base heuristic does not perform well with the percentage of data used. Figures clearly show that all four embedding-based heuristics significantly and consistently perform worse than other heuristics including random.

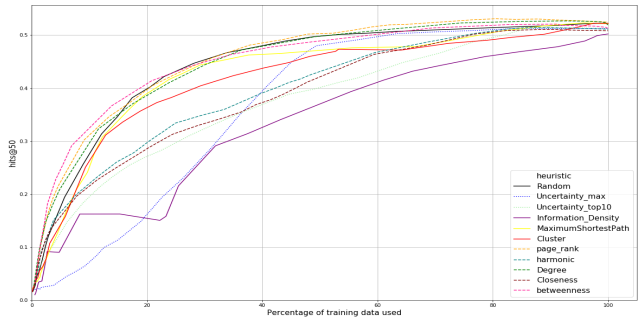
Moving to the WK3l data-sets, figures 5.11 and 5.12 plots the performance with training data percentage for WK3l data-sets. For WK3l-english to German data (Figure 5.11), betweenness significantly out-performs other heuristics. All other heuristics do not significantly out-perform the random heuristics. For WK3l-english to French data (Figure 5.12), betweenness significantly out-performs all heuristics except maximum shortest path distance heuristic for some measurement types. Performance of betweenness and maximum shortest path distance is similar for measures hits@1 and MRR, but for hits@10 and hit@50, betweenness is the clear best performer. Therefore, it is clear that betweenness consistently performs best for WK3l data-sets.



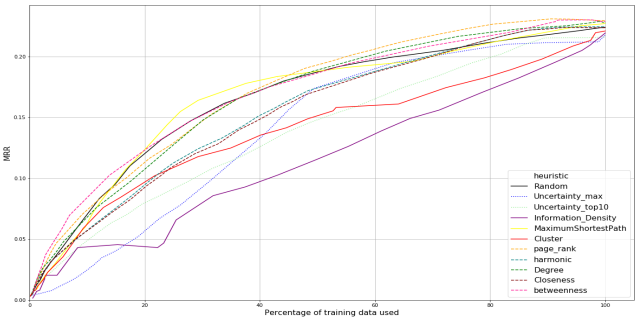
(a) Performance measure- hits@1



(b) Performance measure- hits@10

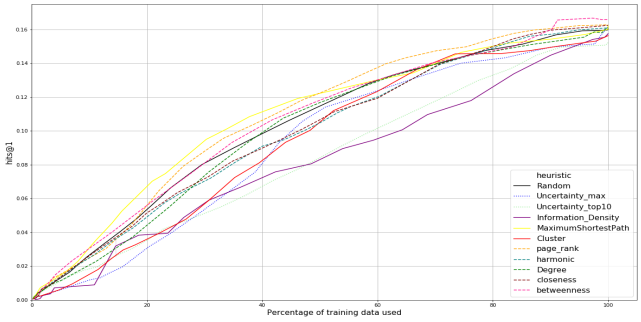


(c) Performance measure- hits@50

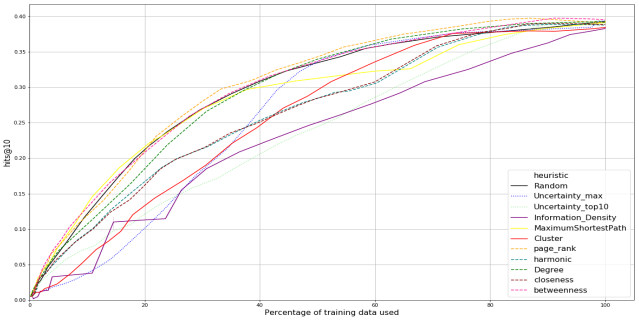


(d) Performance measure- MRR

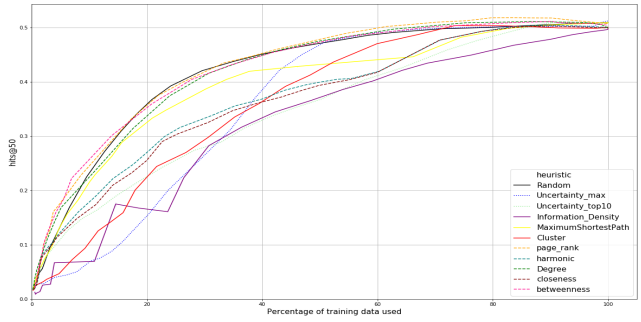
Figure 5.8: Performance with Percentage of training data used DBP15K- French to English



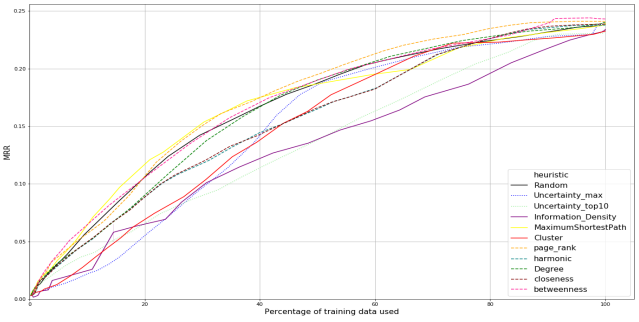
(a) Performance measure- hits@1



(b) Performance measure- hits@10

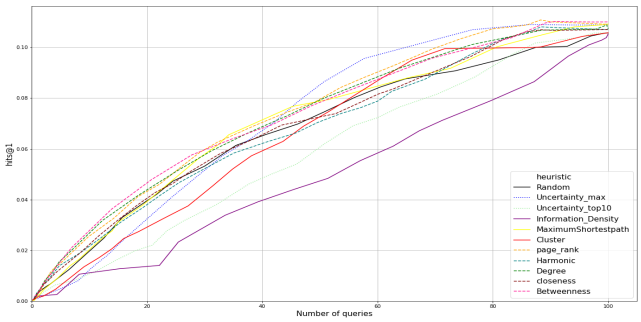


(c) Performance measure- hits@50

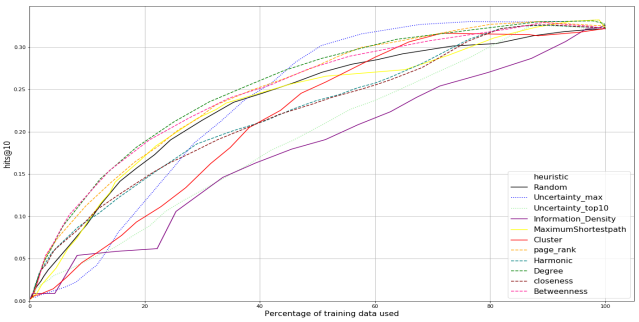


(d) Performance measure- MRR

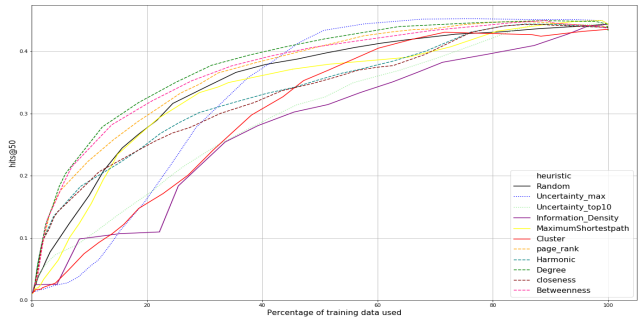
Figure 5.9: Performance with Percentage of training data used DBP15K- Japanese to English



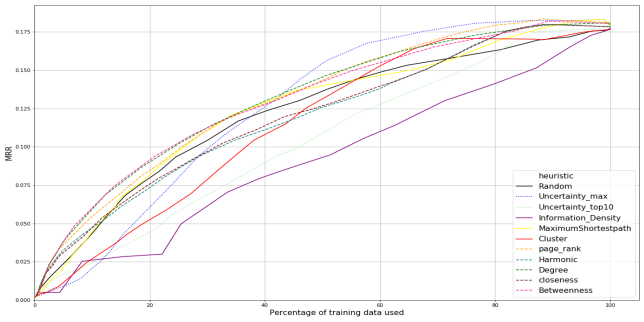
(a) Performance measure- hits@1



(b) Performance measure- hits@10

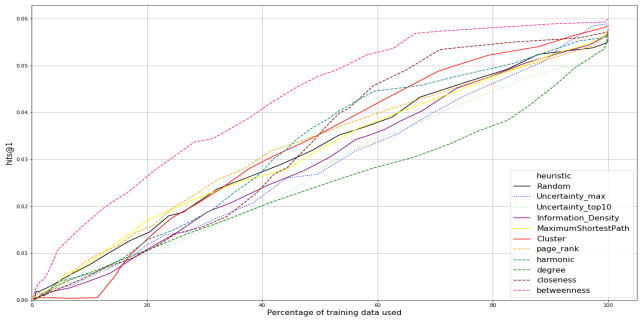


(c) Performance measure- hits@50

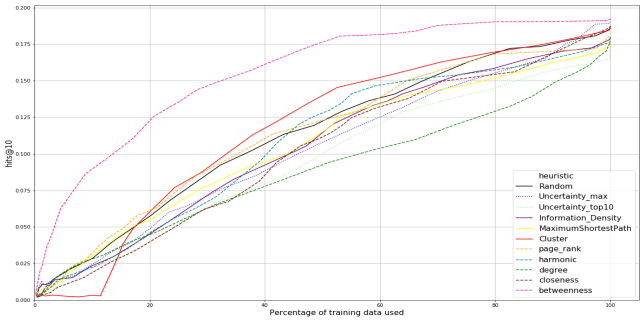


(d) Performance measure- MRR

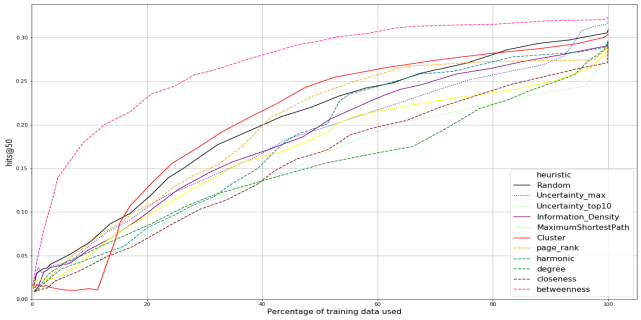
Figure 5.10: Performance with Percentage of training data used DBP15K- Chinese to English



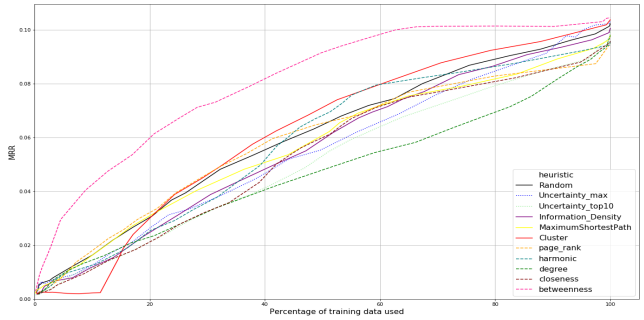
(a) Performance measure- hits@1



(b) Performance measure- hits@10

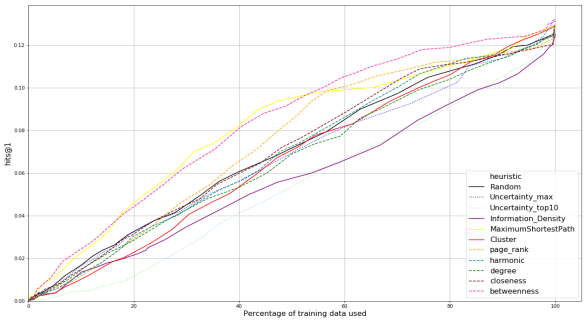


(c) Performance measure- hits@50

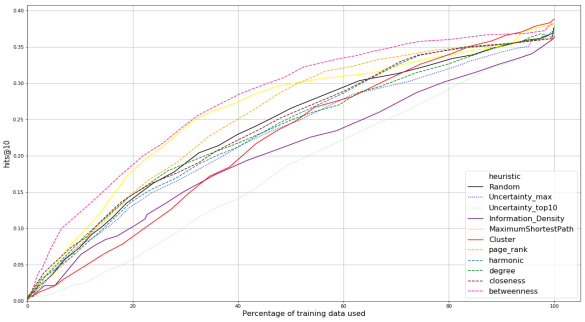


(d) Performance measure- MRR

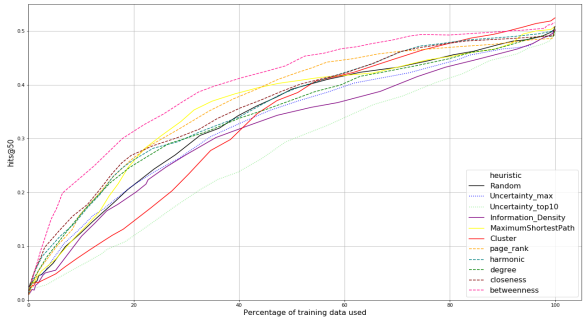
Figure 5.11: Performance with Percentage of training data used WK3l- English to German



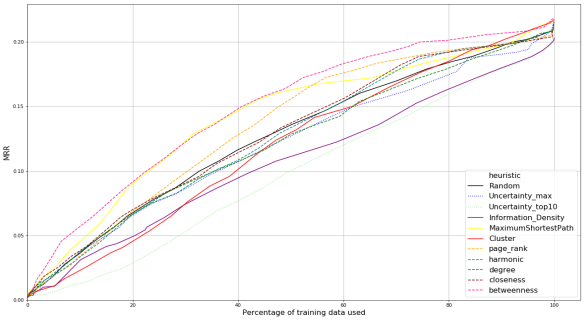
(a) Performance measure- hits@1



(b) Performance measure- hits@10



(c) Performance measure- hits@50



(d) Performance measure- MRR

Figure 5.12: Performance with Percentage of training data used WK3l- English to French

When consider the performance with training data percentage used across both data-sets, betweenness is the only heuristic that consistently performs better than random. Embedding-based heuristics seems to perform very poorly in DBP15K data-sets, however, there seems to have some improvement in performance for these in WK3l data-sets. This bad performance of embedding based heuristic is quite an unexpected result, especially in the case of uncertainty sampling. Aim of uncertainty sampling is to query the instance that contains the highest information, therefore we expect that uncertainty sampling performs well in this setting. The reason for this bad performance maybe because embedding simply does not capture the informativeness of alignment at the initial fitting of the model (because training data is limited). We will dive more into this in the section 6.4.

5.3 Comparison of Performance with Number of Queries and Percentage of Data Used

If we consider the performance with number of queries or percentage of data used, betweenness is the best performing heuristic for both instances. Our main aim was to identify the best performing heuristic with respect to the number of queries, however, comparing performance with respect to the percentage of data used gives unique insights to the performance of each AL heuristic. As

we discussed, to get good performance with number of queries, two factors are required; the high success rate for querying aligned nodes and high information content in the queried aligned nodes. This second aspect was clearly visible in WK3l English to German data-set. If we examine Figure 5.7(a), we can see that the query success of betweenness is not very high. There are multiple AL heuristics that have a higher success rate than the betweenness, but it significantly out-perform all these heuristics when performance is plotted with number of queries (Figure 5.5). The reason for this can be derived from Figure 5.11. It clearly shows that the performance of betweenness with respect to the percentage of data used significantly out-performing all other heuristics including random heuristic which we used as the base for comparison. So the conclusion we can derive from this is that even-though an AL heuristic does not have a higher success rate of finding aligned nodes, if it finds aligned nodes that contain high information, still it can perform well. However, according to results, we should focus more on implementing AL heuristics that have a higher success rate of finding aligned nodes (better than that of random) as it seems that success rate has more impact on performance than informativeness of queried align nodes.

5.4 Performance and Data-sets

When consider the performance of AL heuristics across data-sets, there is an effect on the performance from the data-sets. For all subsets of DBP15K, the performance of all AL heuristics was consistent. AL heuristics betweenness, degree and page rank all performed better than random for all three subsets of DBP15K. However, for WK3l subsets, this was different. For WK3l subset English-German, the only betweenness performs significantly better than random. For WK3l subset English-French, however, have better than the random performance by betweenness, degree, page rank and maximum shortest path distance. The reason for this different performance may be due to the characteristics of the data-sets. If we consider the DBP15k, it is constructed such that align nodes are popular, that is, those nodes have at least 4 relationship triples. In a KG, relationship triple act as an edge, therefore betweenness and degree score of these aligned nodes should be high as these measures highly depend on the number of edges connected to the node. This should be the reason for exceptionally high success rate of finding aligned nodes in betweenness and degree in DBP15K data-sets. In the WK3l English-German data-set, German KG is much sparser than the English. When consider the English-French data-set, French KG is sparser than English KG, but not much as that of German KG in English-German data-set (table 4.4). We can attribute these different sparsity levels for the different performance derived for two subsets of WK3l.

Chapter 6

General Discussion and Conclusions

This chapter gives a basic overview of the study, implementation steps and justification for those steps and a summary of the study findings. Furthermore, we will discuss problems encountered in implementing some AL methods and give some future implementation suggestions in the research area 'Active learning for Entity Alignment'. Finally, we give some principal conclusions derived from the study.

6.1 Overview of the study

The main objective of the study is to evaluate the performance of Active Learning Methods to the task of entity alignment. For this purpose, we applied number of AL heuristics on the Entity Alignment task. Altogether, we implemented ten AL heuristics and evaluate their performance using five data-sets (derived from two main data-sets). We also implemented a random heuristic to use as a basis to evaluate the performance. Data-sets used are cross-lingual where each data-set contains data on two KGs. To align KGs, we use the KG alignment model GCNAlign. The implementation process of the experiments is described in Chapter 4 and the analysis of the data from the experiments is given in Chapter 5. If we dive into the motivation behind the study, the premise is that training data for KG alignment models are hard to come by in most practical situations, hence to overcome this situation AL is used.

When aligning two KGs by an alignment model, training data required would be aligned entity pairs (set of same entities in both KGs) or alignment triples of each KG depending on the alignment model. In real-world KG alignment problems, this data is hard to find. Therefore in a practical situation, research may have to choose some entities from one KG and check for similar entities in other KG to create the training data. So, rather than just randomly choosing some entities to create the training data, the researcher can use AL to identify the entities which will be most suitable to the alignment model. This will enable the researcher to fit the model with little training data, but get the maximum performance from the model within the budget for creating the training data. In this context, our objective was to identify the best AL heuristics so that, researches can effectively use AL methods in their problems. To study this, our approach was to evaluate the performance of each AL heuristic with the number of queries. Also, we studied the performance of each heuristic with percentage of the data used, which enable us to identify the ability of each AL heuristics to query the instance with the most information.

6.2 Discussion of Implementation Steps

Chapter 4 gave a detailed description of the implementation steps. Here we will mostly focus on the AL implementations used in the study. For choosing an instance from AL heuristic, the first step is to rank all the available nodes in both KGs. We can consider this as the pool-based setting in AL implementation. To rank nodes, we will pool the nodes in both KGs into one pool. Then assign a score based on the AL method. In every AL heuristic considered in the study, a higher score indicated lower ranking, which means higher priority when querying. This score for ranking was calculated for nodes in each KG separately. For example, when calculating the centrality scores of a node, we only consider the KG that node is part-of. Since in each data-set, the number of entities of both KGs is similar we did not adjust AL scores derived for nodes of each KG according to KG size. Another factor we had to consider is the changing of AL scores according to the step size (defined in section 4.3.3). In centrality-based AL heuristics we will calculate the centrality of each node, and order(or rank) the nodes before querying. We do not change this order throughout our implementation process. But for all embedding based AL heuristics, this ordering will be changed after each step size. This is because, after each step-size, we refit the GCNAlign model by adding new alignment data to the training set. Therefore we will have a new embedding after each step size. This change of ordering is applicable for the maximum shortest path distance (MSPD) heuristic also. MSPD ordering changes after every step size because the source list (which consists of all aligned nodes up-to-the last step size considered) given to calculate the shortest path distances changes after each step size. Therefore, the implementation of AL heuristic was done in batches. That is, once we identify the number of queries according to step size (number of queries is the difference between last and current step size), we will order all the available nodes according to AL heuristic and select the nodes from the rank one to number of queries. Then all these nodes are provided to oracle and oracle will label all these nodes one by one.

When implementing AL heuristics, we had to decide on ways to implement certain steps. In the first uncertainty sampling implementation, to calculate the entropy score of a node we use the maximum of its' similarity with nodes from other KG. By taking maximum instead of mean or median, we make sure that the entropy score is calculated based on the nodes' most likely alignment node in other KG. This is because, higher the similarity between two nodes, higher the probability that these nodes are aligned. In the second uncertainty sampling implementation, rather than taking only the maximum similarity, we took the top ten similarities. This would give more representation of entropy distribution rather than depending on only one value like the last implementation. The reason to use the top ten rather than other number of similarities is that we tried some other numbers as well. But the results indicate a same or worse performance. However, we only tried a couple of values such as 50 and 100 and we did not comprehensively study the effect of the number. However, considering the results of the study, it is reasonable to assume that this number would not have a significant impact on results from uncertainty sampling implementation.

In the implementation of Cluster-based active learning, it was very hard to determine the number of clusters to run K-means. For our implementation, we used the square root of the number of available nodes in each KG as the number of clusters (recall that we run K-means separately on two KGs). The use of this number of clusters was inspired by the paper [23]. The authors of the paper set maximum number of clusters as the square root of the number of instances in their algorithm to find optimal number of clusters. Hence we use that number as it is the upper bound of the optimal number of clusters. We did not try to apply the algorithm in the paper as we want to keep our method simple and the application would be very hard in our setting.

6.3 Summary of study findings

The main finding of our study is that betweenness heuristic is the only AL heuristic to consistently and significantly out-perform random heuristic. Further, AL heuristics degree and page rank did out-perform random for most of the data-sets considered in the study. All the other AL heuristics considered did not consistently and significantly outperform random. Furthermore, we identified that the two main factors that AL heuristic performance depends on; the ability of the AL heuristic to find aligned nodes and the ability to find aligned nodes that contain more information about alignment between two KGs. To identify the AL heuristics ability to find aligned node with high information, we plotted the performance of heuristics with the percentage of data used. For this situation also, betweenness is the only heuristic that consistently out-perform random. In addition, we concluded that among these two factors heuristic performance depends on, the factor we must most focus should be the ability to find aligned nodes.

6.4 Discussion of the results

As pointed out in Chapter 5, our analysis gave some surprising or unexpected results. For both uncertainty sampling implementations, which specially focused on information content, section 5.3 shows that it does prioritize the alignment pairs that have the most information. In fact, all the AL heuristics that were based on embeddings do not perform well with respect to both the number of queries and the percentage of data used. The reason for this may be, to learn reasonable representation for embeddings, a decent amount of training data is required. We expected that all the embedding based AL heuristics to not-perform very well at initial step sizes. However, we expected that the performance will pick up quickly once a reasonable amount of training data is given to the model. But this is not the case as in most situations considered as performance of all embedding based AL heuristics took a large number of queries or large data percentage to pick up and perform on-par with other AL heuristics. A possible reason for this may be the characteristics of data-sets. As shown in Chapter 5, we saw that embedding based AL heuristics performs much better for WK3l data sets than DBP15K data sets. Another reason for this may be the alignment model we used. We did not test this theory as we only did the experiments with only the GCNAlign model. Hence this reason is open as a possible explanation for this performance. Finally, when moving to the best performing heuristic betweenness, it is a centrality measure that can be considered as a measure of the bridgeness of a node. That is, it measures how often a node is a bridge (connection) between other nodes. If we consider the entity alignment task, the main aim is to connect(bridge) two (or more) small KGs into one large KG. For this, we use common entities between two small KGs. The large KG will consist of all nodes and the relation of two small KGs. Since these common entities are working as bridges between nodes in two KGs, these common entities should have high betweenness values in the large KG. Then we can argue, nodes that have high betweenness values in small KGs are more likely to be common entities. Therefore, we can see this as a good explanation for the high performance of betweenness in entity alignment task.

6.5 Limitations

In the study, we implemented ten AL heuristics. However, we only used two areas of AL query strategies out of the four main areas of query strategies discussed in Chapter 2. We did not implement AL heuristic based on reducing of hypothesis space and minimizing of expected error and variance. Under the reduction of hypothesis space, the most used method is query by committee (QBC). To

implement QBC in our setting we may need to implement multiple alignment models and have to define a method to calculate vote entropy as defined in equation 2.4. Therefore implementing some version of QBC can be complex and expensive because of the use of multiple alignment models. To implement the expected error/variance reduction AL, we require the probability distribution of oracles' labeling and models' probability distribution of making an error. These requirements are simply not available in our setting. Another limitation in our setup is that if a selected node does not have an alignment in other KG, we do not use it in the model. This is a major difference from usual AL implementations where instances are given a possible label by the oracle and all the instances that are labeled are used in the model fitting.

6.6 Future Implementations

All the AL heuristics implemented are basics implementations of the AL strategies in literature. We can suggest some improvements AL heuristics for the current implementations.

- For Cluster-based AL, instead of K-means clustering, we can use hierarchical clustering for AL heuristic as given in this paper [25]. This method first cluster the data with hierarchical clustering. Then, rather than selecting an instance, the algorithm selects a cluster (randomly or proportional to cluster size) and randomly selects an instance within the cluster. Once that instance is labeled, the clustering structure is re-structured according to the label of the instance. This algorithm can be used to avoid the problems in cluster-based AL which was discussed in Chapter [25].
- Use combination of AL heuristics to define a new AL heuristic as mentioned in the paper [22]. In the paper, the AL strategy which consists of uncertainty sampling, centrality, and density is implemented on graph data.
- Implement a AL heuristic based on QBC as given in the paper [26]. Paper discusses ways to obtain ensembles of Deep Learning models for AL.

Furthermore, in our implementation of the model, we do not use the nodes that do not have an alignment. However, we can use these nodes to modify the connectivity matrix (defined in Chapter 2), which represent edges of KGs after each step size. That is, we can use the exclusive nodes (nodes that do not have an alignment) to modify the edges of the KGs so that the information contained in those nodes would be used in the model.

6.7 Conclusions

By considering all findings of the study, following principal conclusions can be made.

1. Betweenness consistently and significantly out-perform all other heuristics considered in the study.
2. Degree and Page Rank heuristics also performs well (better than random), but not consistently.
3. Centrality-based AL heuristics performed better than embedding (or model) based AL heuristics.
4. Query success rate (rate of finding matching nodes) is one of the most important factor for the performance of heuristic.

5. Information content in the queried aligned nodes is also an important factor in the performance of heuristic. But not as much as query success.

Bibliography

[1] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[2] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[3] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706, 2007.

[4] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[5] Yanchao Hao, Yuanzhe Zhang, Shizhu He, Kang Liu, and Jun Zhao. A joint embedding method for entity alignment of knowledge bases. In *China Conference on Knowledge Graph and Semantic Computing*, pages 3–14. Springer, 2016.

[6] Muhao Chen, Yingtao Tian, Mohan Yang, and Carlo Zaniolo. Multilingual knowledge graph embeddings for cross-lingual knowledge alignment. *arXiv preprint arXiv:1611.03954*, 2016.

[7] Zequn Sun, Wei Hu, and Chengkai Li. Cross-lingual entity alignment via joint attribute-preserving embedding. In *International Semantic Web Conference*, pages 628–644. Springer, 2017.

[8] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 349–357, 2018.

[9] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[10] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes—a time-efficient approach for large-scale link discovery on the web of data. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[11] François Scharffe, Yanbin Liu, and Chuguang Zhou. Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*, 2009.

[12] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk-a link discovery framework for the web of data. *Ldow*, 538:53, 2009.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[14] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[15] Tom M Mitchell. Generalization as search. *Artificial intelligence*, 18(2):203–226, 1982.

[16] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[17] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[18] Jaeho Kang, Kwang Ryel Ryu, and Hyuk-Chul Kwon. Using cluster-based sampling to select initial training set for active learning in text classification. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 384–388. Springer, 2004.

[19] Hieu T Nguyen and Arnold Smeulders. Active learning using pre-clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 79, 2004.

[20] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[21] Max Berrendorf, Evgeniy Faerman, Valentyn Melnychuk, Volker Tresp, and Thomas Seidl. Knowledge graph entity alignment with graph convolutional networks: Lessons learned. *arXiv preprint arXiv:1911.08342*, 2019.

[22] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. Active learning for graph embedding. *arXiv preprint arXiv:1705.05085*, 2017.

[23] Siddheswar Ray and Rose H Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques*, pages 137–143. Calcutta, India, 1999.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Sanjoy Dasgupta and Daniel Hsu. Hierarchical sampling for active learning. In *Proceedings of the 25th international conference on Machine learning*, pages 208–215, 2008.

[26] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. The power of ensembles for active learning in image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9368–9377, 2018.