Alignment of Knowledge Graphs based on Learning to Rank techniques

Victor Eiti Yamamoto　　*Julio Cesar dos Reis*

Relatório Técnico - IC-PFG-20-03
Projeto Final de Graduação
2020 - Agosto

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Alignment of Knowledge Graphs based on Learning to Rank techniques

Victor Eiti Yamamoto
Julio Cesar dos Reis
Institute of Computing, University of Campinas, Campinas, SP, Brazil.

2020

Abstract

Knowledge graphs (KGs) define facts expressed as triples considering subject, predicate and object in the representation of knowledge. Usually, several knowledge graphs are published in a given domain. It is relevant to create alignments both for classes that model concepts and between instances of those classes defined in different knowledge graphs. In this work, we study techniques for aligning entities expressed in KGs. Our solution explores supervised ranking aggregation method in the alignment based on similarity values. Our experiments rely on the dataset from the *Ontology Alignment Evaluation Initiative* to evaluate the proposed method in experimental analyzes.

1 Introduction

The term knowledge graphs was coined by Google when it introduced this technology as the basis for a new web search strategy in 2012 [32]. A traditional search method uses keywords to search for the expected results, but the terms can be ambiguous and limit the retrieved information. The use of knowledge graphs allows the search to be carried out for objects that represent real entities such as places, people and movies. These entities and their relationships allow performing information retrieval by using a context in which the term is searched for. This helps in reducing the ambiguity of terms, and improving the quality of information returned when using the entities' relationships [34].

Large-scale knowledge graphs (KGs) like *DBpedia* [1], *YAGO*[2] and *Wikidata* [3] play a central role as a source of general knowledge. These KGs present a good coverage regarding the entities represented and expressed in several domains. However, they lack covering specific topics or they usually are addressed with little detail [18]. The aforementioned KGs present similar information, as they all use *Wikipedia* as the basis for creating entities and their relationships. Their use in a combined way requires creating links between entities

[1]`http://dbpedia.org`
[2]`https://yago-knowledge.org`
[3]`https://www.wikidata.org/wiki/Wikidata:Main_Page`

of different KGs [33]. For example, "*Black Panther*" [4] is an entity from Marvel Cinematic Universe Wiki, which is mapped to the Marvel Database entity "*Black Panther*" [5].

Hofman *et al.* [18] used a two-step method to create mappings between KGs generated from Wikis. First, mappings were generated between each Wiki and DBpedia. Using these mappings, the KGs were grouped in blocks and the mappings between Wikis were created only between graphs of the same group. To obtain the related entities, a string distance algorithm was used based on the labels of the entities.

Learning to Rank is a machine learning technique for training ordering models. This technique can be used in several areas such as information retrieval, natural language processing and data mining. An example of application of such technique is the retrieval of documents. A system manages a set of documents and when a query is executed, the system searches for documents containing the queried terms, order the documents based on different processed ranking lists and returns the best results [28].

In the creation of an alignment (process of generating mappings between KG entities), the simplest approach might look for all possible pairs in a set. However, this approach becomes prohibitive for larger sets. Locality-sensitive Hashing (LSH) [26] allows the comparison between similar pairs. The items are provided in a hash, in which the probability of two items having the same hash is based on the similarity value between them. In this sense, the candidate pairs are those that are in the same hash bucket .

In this work, we propose and evaluate a method for the alignment of KGs based on Learning to Rank techniques. In our approach, we investigate the candidate reduction methods based on Locality-sensitive Hashing. We experimentally evaluate our proposal based on the test dataset offered by the OAEI (*Ontology Alignment Evaluation Initiative*) competition [6]. The quality of the mappings created by our solution are evaluated by comparing them with the alignment provided by the competition (as a gold standard).

In our results, the proposed technique obtained high recall in the created mappings, but affected precision in several cases. Compared to existing baseline systems, our approach presents lower f-measure for class and instance, but higher f-measure for property.

The remaining part of this work is organized as follows: Section 2 presents a literature review on methods for KG alignment. Section 3 introduces the necessary formalization; Section 4 describes our proposal thoroughly; Section 5 presents the experimental results conducted; Section 6 discusses the achieved results; Section 7 summarizes the conclusions.

2 Related Work

Literature provides several definitions for knowledge graphs. Ehrlinger and Wöß [13] defined knowledge graph as a knowledge base with extended requirement to reasoning. Using this definition, a system must have a knowledge base (*e.g.*, Ontology) and a reasoning engine to generate new knowledge and integrates one or more information sources to be considered as knowledge graphs. DBkWik [18] uses knowledge graphs as synonym of knowledge base

[4]`https://marvelcinematicuniverse.fandom.com/wiki/Black_Panther`
[5]`https://marvel.fandom.com/wiki/Black_Panther`
[6]http://oaei.ontologymatching.org

and ontology like DBpedia [25], YAGO [35] and Wikidata [29]. Hogan *et al.* [21] defines knowledge graph as a graph of data intended to accumulate and convey real-world knowledge, whose nodes represent entities of interest and whose edges represent relations between these entities.

Several systems have been proposed with the creating mappings between entities of KGs. AgreementMakerLight Ontology Matching System (AML) [14] is a framework that performs the mapping between ontologies using four types of matchers and implements a ranked selector. The matchers are lexicon; lexicon mediated by a third ontology; words using the Jaccard index [22]; and a set of parameters exploring string similarity methods. To obtain the best mappings, the results of the matchers are ordered in a unified list from the best to the worst. The mappings are created based on such ordered list.

FCAMap-KG system [7] uses the analysis of formal concepts to create mappings. This system organizes the process into three stages: lexical match, structural match and match filter. In the first step, it creates three formal contexts based on keys for classes, properties and instances. In OAEI KG context, those types were created to fuse different KGs into one coherent KG. Schema type (classes and properties) derives from wikis' constructs and instance type derives from pages about real-world entities [20]. A mapping is created when a formal concept contains objects from the two KGs under alignment. In the second stage, the previously obtained mappings are used to create a structured formal context. In this step, the focus is on creating mappings between the instances using RDF triples whose properties and subsequent instances were already mapped. In the last step, the mappings are selected so that each entity has only one mapping. This operation can be carried out because the OAEI 2019 KG competition uses only 1:1 matches. If an entity has more than one mapping, the mapping with more structural attributes and lexical keys in common is selected in this technique.

DOME system (Deep Ontology MatchEr) [19] uses doc2vec method to obtain the mappings. Doc2vec is an algorithm that learns fixed-length feature representations from variable-length piece of text. Each document is represented as a vector which is trained to predict words in the documents [24]. This system is organized into five stages: String matching; confidence adjustment; instance-based class matching; type filter; and cardinality filter. In the first step, the text is divided into tokens and a series of string matching methods is used to obtain the mappings. In the second stage, the system goes through all correspondences and assigns a new confidence using the doc2vec method. In the third step, new mappings are created for classes based on instances that were already mapped. The idea is that if instances of a class present mappings, then there is a greater chance that a class migth also have a mapping. In the fourth step, the mappings are filtered so that only mappings between entities of the same type remain. In the last step, the mappings are filtered to obtain only one mapping per entity.

Learning to Rank (LTR) is a machine learning technique for training the model to rank [28]. The system manages a system of documents. When queried, the system retrieves documents related to the query, rank the documents and returns the top ranked documents. The differences between learning to rank and other models are that LTR does not need to predict absolute value of the items (regression); it does not need to predict the class of items (classification); the important thing is to obtain the relative ranking of items.

There are three common approaches to learning to rank: pointwise, pairwise and listwise. Pointwise is based on regression and classification. The general criteria is to create a ranking function f that learns to assign an absolute score to each item in isolation. The objective is to minimize the cost function based on the absolute gold score. This approach is generally more difficult than necessary. Pairwise approach is based on rank-preference model. The general criteria is to find a ranking function f that learns to rank pairs of items. The objective is to minimize the cost function of misclassified pairs. To solve this problem, any binary classifier can be used. The listwise approach is based on list of documents and aims to optimise the most appropriate task metric [10]. Our method explored *LambdaMart* which is a pairwise learning to rank technique. Destro *et al.* explored several rank aggregation techniques for aligning crosslingual ontologies and found that *LambdaMart* have the best results [23]. *LambdaMart* also won *Yahoo! Learning To Rank Challenge* [8]. *LambdaMart* is a technique that combines *MART* and *LambdaRank* techniques and so those two techniques is explained below separately.

MART (Multiple Additive Regression Trees) [15] is a boosting paradigm based on the gradient method for additive expansion used in regression trees. Boosting is a technique based on PAC (probably approximately correct) and its main focus is on learning efficiency. The idea is to create a series of weak classifiers that together form a strong classifier. This method starts with an original training set made up of N different examples. Each element in the original test set is given a weight proportional to the probability of appearing in the filtered test set. Initially, each element is given an equal unit weight so that they all appear in the initial set. At each step, the weight is changed and some elements can have multiple entries in the filtered test set. The tree is trained using the filtered training set. The original test set is used to assess the quality of the tree's classification. The error is used to update the weight of each element in the test set. The training process is repeated until the result is stable [12].

LambdaRank [5] is a pairwise learning to rank technique that uses only the gradient of the cost to reduce the error. Previous approaches like *RankNet* [4] use cross entropy cost function that penalizes the number of inversions in output ranking to get the correct order for all element in the list and use stochastic gradient descent to optimize results. *LambdaRank* uses ranking quality measures like Normalized Discounted Cumulative Gain (NDCG) that handle multiple levels of relevance and a position dependence for results. This measure gives more weight for higher ranked results, so the optimization of those measure is more focused on top results. Both techniques use gradient descent, but *RankNet* uses the number of pairwise error and *LambdaRank* uses weighted ranking quality measure.

LambdaMart is a version of *LambdaRank* using boosted tree combined with the *MART* paradigm. The difference between *LambdaMart* and *LambdaRank* is that *LambdaRank* updates its weights for each test entry queried, while *LambdaMart* updates only a few parameters, but using all test set. This means that *LambdaMart* can reduce the quality of the result for some queries if the overall quality is higher.

Cruz *et al.* [9] explores learning to rank to deal with unbalanced test set. This technique was explored in other unbalanced class problem such as link prediction [27]. Our study explores learning to rank techniques in the creation of mappings between different types of entities expressed in KGs.

3 Preliminaries

This section provides a series of formal definitions relevant to our solution.

Knowledge Graph. A knowledge graph $\mathcal{K}$ accumulates and conveys knowledge in term of entity and relations [21]. Formally, a knowledge graph $\mathcal{K} = (\mathcal{E}_{\mathcal{K}}, \mathcal{R}_{\mathcal{K}})$ consists of a set of entities $\mathcal{E}_{\mathcal{K}}$ represented as nodes; and entities are interrelated by directed relationships $\mathcal{R}_{\mathcal{K}}$. Each entity $e \in \mathcal{E}_{\mathcal{K}}$ has a unique identifier and type. The entity identifier uses Universal Resource Identifier (URI) that is a string used to identify resources and provide a mean of locating the resource [1]. Entity type can be of schema level, formed by properties and classes, and instance level. Each relationship $r(e_1, e_2, e_3) \in \mathcal{R}_{\mathcal{K}}$ is a triple consisting of a subject, a predicate and an object.

Mapping. Given two entities e_s and e_t from two different KGs, a mapping m_{st} is defined as:

$$m_{st} = (e_s, e_t, conf) \tag{1}$$

The $conf$ is the similarity value between e_s and e_t indicating the confidence of their relation. We define $\mathcal{M}^j_{ST}$ as a set of mappings m_{st} between two KGs $\mathcal{K}_S$ and $\mathcal{K}_T$.

Similarity. Given two entities e_i and e_j, the similarity between them is defined as a function that calculates the similarity score between them returning a real value in the interval [0,1]. The function can explore different techniques for the similarity computation, like string-based processing and semantic-based techniques which explore background knowledge for similarity computation. Formally:

$$sim(e_i, e_j) = f(e_i, e_j) \tag{2}$$

Rank Aggregation. Consider a set of rankings R_1, R_2, R_3, ..., R_n formed by set of similarity proposals S_1, S_2, S_3, ... S_m. Each similarity list S_k requires a different function $sim(e_i, e_j)$ with a source e_i and target e_j. Each $s \in S$ is defined as a triple $s = (e_i, e_j, sim(e_i, e_j))$ and the entity e_i can be used to retrieve a set of similarity. Rank Aggregation produces one single rank that aggregates all given ranking. Equation 3 is used to include new set of ranking to aggregate. Figure 1 shows an example of rank aggregation. Each column represents a ranking and in each rank the yellow box represents an element that is ranked in different positions for each rank. An aggregated rank is created and used to map entities in our solution.

$$rankAggregation.aggregate(R) = R_{agg} \bigcup R \tag{3}$$

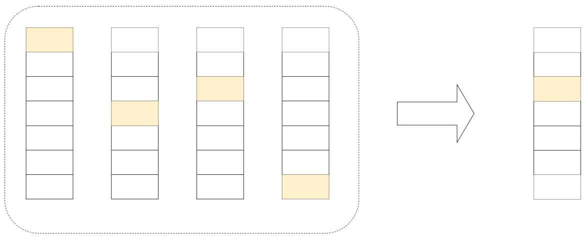


Figure 1: Rank Aggregation

Given an aggregated rank *rankAggregation* and an entity e, it is possible to query the aggregated rank presenting e. Equation 4 represents a function that returns a ranking formed by set of similarity that have input e as query filter.

$$rankAggregation.query(e) = R, s \in R_{agg}, s.id = e \tag{4}$$

Locality-sensitive Hashing. An entity e is converted to a set of n-grams and added to a bucket of the hash. The equation 5 describes a function that insert an entity e to the hash H. Locality-sensitive Hash is an algorithm that uses random projections to construct hash codes which pairwise distance is preserved when the length of codes is sufficiently large [36]. Given an entity e and a locality-sensitive hash LSH, entity e can be used to query a similar entity inside such hash based on a similarity method. The equation 6 describes a function that query for similar entities compared to entity e_i and returns a set of entity E where $e_j \in E$ have similarity $sim(e_i, e_j)$ greater than a given threshold τ [26].

$$LSH.insert(e_i) = H \bigcup Hash(e_i) \tag{5}$$

$$LSH.query(e_i) = E, e_j \in E, sim(e_i, e_j) > \tau \tag{6}$$

Problem statement. Given two KGs $\mathcal{K}_{\mathcal{S}}$ and $\mathcal{K}_{\mathcal{T}}$, the problem addressed in this work is to obtain all mappings between entities from these KGs by using rank aggregation techniques and reduce explored candidates with the use of Locality-sensitive Hashing.

4 Our proposal to link Knowledge Graphs

Our goal is to create appropriate mapping for each entity of a source KG $\mathcal{K}_{\mathcal{S}}$ to a target KG $\mathcal{K}_{\mathcal{T}}$. Figure 2 shows our defined workflow for this purpose. It is organized in four main steps: input process, candidate pairing, similarity calculation and map creation.

In the first step, both KGs are processed and their entities are divided by their type (class; property; instance). Every triple formed by subject, predicate and object is extracted if the predicate is a RDFS label. The Resource Description Framework (RDF) is a framework for representing information in the Web [11] and RDFS is the schema used to model RDF data. RDFS:label is an instance that provides a human-readable resource's name [2]. If the subject of the triple is an URI, a new entity is created and it is identified by the extracted URI. The entity type is defined by analysing if the type name is contained in the URI. Class, property and instance is identified by class, property and resource, respectively. For example, `http://dbkwik.webdatacommons.org/starwars.wikia.com/class/character` is an URI that identifies a class type entity; and `http://dbkwik.webdatacommons.org/starwars.wikia.com/resource/Anakin_Skywalker` is an URI that identifies an instance type entity.

In the second step (candidate pairing), the method creates the list of candidate entities for each entity from the source KG $\mathcal{K_S}$. For schema types (class and property), all target candidates are considered as candidates. For instance type, locality-sensitive hashing is used to create the list of candidate entities (cf. Subsection 4.1). All entities from target KG $\mathcal{K_T}$ are inserted to the hash $\mathcal{H}$. After this process, each entity e from source KG $\mathcal{K_S}$ query hash $\mathcal{H}$ to retrieve the set of candidate entities $\mathcal{E}$.

In the third step (similarity calculation), each pair of source entity and candidate entity similarity is calculated. The similarity is calculated using entity name from URI. URI is formed by three main components: scheme, authority and path. Only the path after the type identifier is used to calculate the similarity. For example, `http://dbkwik.webdatacommons.org/starwars.wikia.com/resource/Anakin_Skywalker` has `http` as scheme; `dbkwik.webdatacommons.org` as authority; and `starwars.wikia.com/resource/Anakin_Skywalker` as path. The entity name used to calculate the similarity is `Anakin_Skywalker`. Our solution explored four different methods: Levenshtein, Jaro, Babelnet and Wordnet (cf. Subsection 4.3 for the explored similarity methods).

In the fourth step (mapping creation), similarity values are aggregated by using the *LambdaMart* method. Our solution generates the final classification (pair of entities expressing the mapping). Each source KG entity receives an alignment with the candidate entity with the best classification. Alignments are filtered to remove multiple alignments to the same entity from the target KG and below than a threshold (cf. Subsection 4.1).

4.1 Locality-sensitive Hashing

Broder [3] describes a min hash technique to identify almost duplicate web documents. MinHash is an implementation of locality-sensitive hashing used to calculate a hash that maintain similarity property. The proposal is that each document has a sketch of a few hundred bytes consisting of a collection of shingle. Each shingle is formed by a continuous substring of words that is converted into a unique numeric identifier based on polynomial arithmetic of a given size. The size of the identifier defines the probability of collision between documents. The document is then identified as a sequence of identifiers and the similarity between two documents is given by the Jaccard distance of identifiers.

Figure 3 shows how min hash technique is used with an locality-sensitive hashing. In this example, there are three entities described with three shingles. The set of entities is converted to a matrix called characteristic matrix (A in figure 3). The characteristic matrix

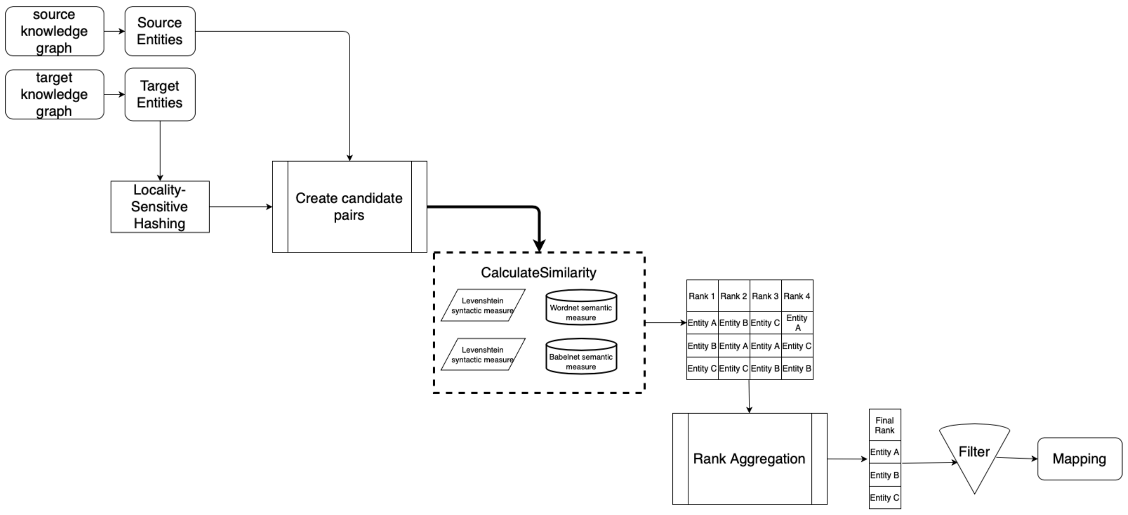


Figure 2: Our mapping technique workflow

represents the presence of certain shingle inside each entity. It is 1 if the entity have the shingle and 0, otherwise. The characteristic matrix is converted to a signature matrix using min hashing (B in figure 3). First, random permutations of row index is created (represented by α, β, γ). Second, the characteristic matrix is iterated using the permuted order and the number of the permutation used to find the first 1 in the column is recorded in the signature matrix. Each permutation fills a row inside signature matrix.

Figure 4 represents the step by step process to convert characteristic matrix to signature matrix. In the example, first permutation is 1, 2, 3 (permutation α). The algorithm iterates each row using permutation order. The first row is explored and the columns that have value 1 are column 1 and column 3. In the signature matrix, the first row receives 1 in the columns 1 and 3 (A in Figure 4). Continue the iteration through permutation. The second row is examined and only column 3 has value 1. Only the first time 1 is found in the column is recorded, so this entry is not recorded in the signature matrix (B in figure 4). The last iteration examined the third row and columns 2 and 3 have value 1. The column 3 is ignored again and column 2 doesn't have any value in the signature matrix, so the first row in second column is filled with value 3 (C in 4). The next rows in the signature matrix are filled with the permutations β and γ.

The last step is the creation of locality-sensitive hashing (C in Figure 3). The signature matrix rows are divided in bands with same row size. Each signature part inside the band is hashed and put in a bucket. Two entities, which creates at least one band signature creating the same hash has more probability to be similar. In the example, signatures are divided in three parts. In the first band, signatures 1 and 3 have same hash result, represented as columns s1 and s3 in the signature matrix. This means that they are in the same bucket for at least one bucket.

All entities in the target KG $\mathcal{K}_\mathcal{T}$ are recorded in the respective buckets. The entities

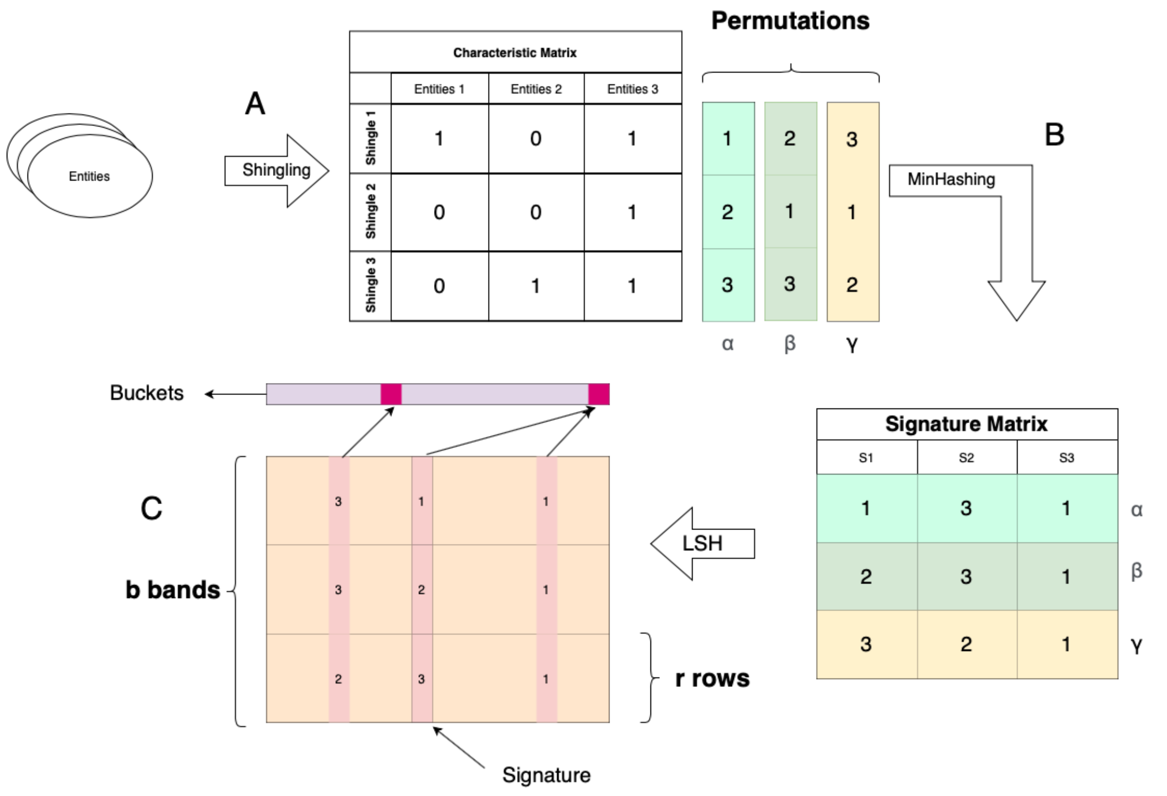


Figure 3: Locality-sensitive Hashing (Adapted from Gupta [16])

from the source KG $\mathcal{K}_\mathcal{S}$ calculate hash in the same way, but they are not recorded. In this case, the calculated hashes are used only to retrieve all entities in the same bucket.

The size of bands affects the chance of entities sharing the same bucket. If more bands are created, there is less rows inside each band that leads to decrease possible buckets results. It means it creates more false positive cases. If less bands are created than it creates more false negative cases. This property can be used to set an approximated threshold. If the similarity between two entities is greater than threshold, they are similar [16] [31]. Given n that is the number of permutations and signature length, b is the number of bands, r is the number of rows in each band and τ is the threshold. Equation 7 and 8 are used to decide the number of bands and rows for a fixed number of permutation and threshold.

$$br = n \tag{7}$$

$$\tau \sim (\frac{1}{b})^{(\frac{1}{r})} \tag{8}$$

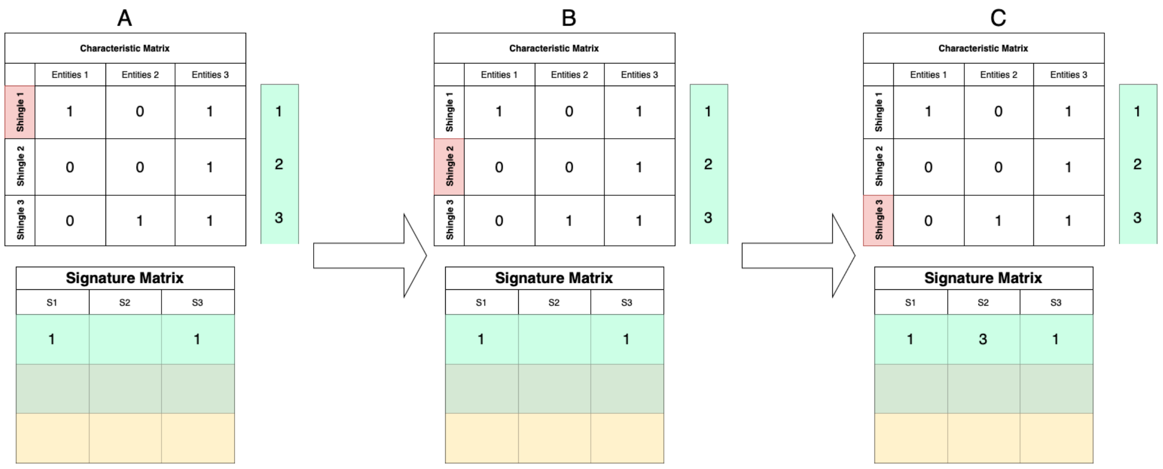


Figure 4: Process to convert Characteristic Matrix to Signature Matrix

4.2 Learning to Rank for the alignment of KGs

Figure 5 presents how learning to rank is applied to our study context. First, the rank aggregation model is created using a training set (cf. A in Figure 5). Second, the pair of source KG entity e_s and the target KG entity e_t have their similarity calculated for each similarity method sim (cf. Subsection 4.3) and the triple (e_s, e_t, sim) is recorded as an entry for the system (cf. B in Figure 5). Third, each entity of the source KG is used as a query to retrieve all similarity entries that form different ranks for each similarity method (cf. C in Figure 5). In the last step, retrieved ranks are aggregated using the trained model and return one rank as result (cf. D in Figure 5).

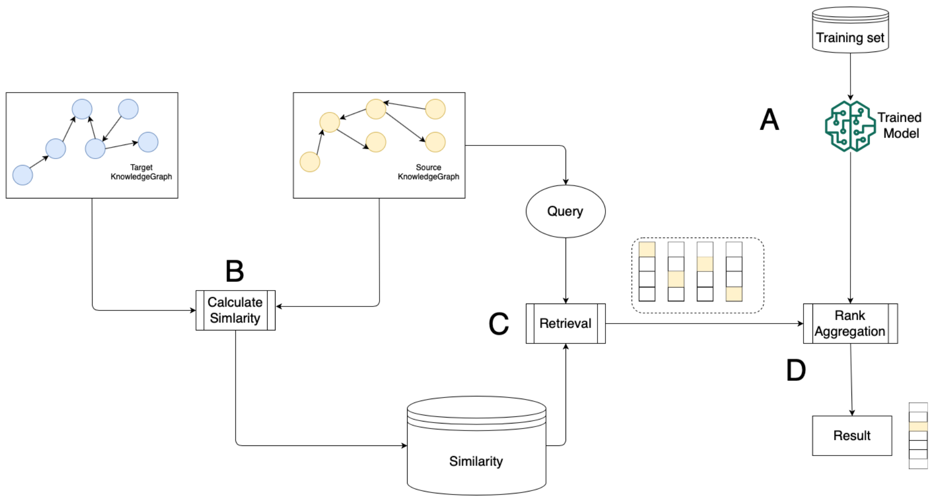


Figure 5: Learning to Rank technique applied to our approach (based on [10])

Learning to Rank techniques like *lambdaMart* [5] are supervised learning tasks. It needs training and testing phases. In the training phase, a model is created to be used later to aggregate ranks (cf. A in figure 5). The training data is consisted of queries set Q as a set of documents to be retrieved D and a set of possible labels Y. The training set S is formed by triples (q_i, d_j, y_{ij}), where $q \in Q$, $d \in D$ and $y \in Y$. The set $Y = 1, 2, ..., n; n \in N$ is the relevance of the document d_i for the query q_j. The process to put relevance to the pair query and document is analogous to the labeling in other techniques.

In our approach, a set of queries Q is formed by URI from source KG's entities; the set of retrieved documents D is formed by candidate entities from target KG's entities; and the set of labels Y receives a value according to the presence of the pair in the gold standard. If the pair (e_i, e_j) is in the gold standard, the triple (q_i, d_j, y_{ij}) receives values $(e_i, e_j, 1)$; and the triple receives value $(e_i, e_j, 0)$ if not present.

The training model creates an ranking model $f(q, d) = f(x)$ that assigns a score to a given pair query and document. For a set, the training model creates a ranking model $F(q, D) = F(X)$ that returns a list of scores that can be converted to a ranking of documents using the score from $f(q, d)$ to sort the documents D [28]. In our approach, the model creates a ranking model $F(e_i, C) = F(X)$, where e_i is an entity from source KG and C is the set of candidate entities from target KG. This model is used to sort all candidate entities for a certain entity from source KG.

4.3 Similarity Techniques

Our proposed technique explores four method for similarity computation to generate rankings to be aggregated.

- Levenshtein similarity, also known as edit distance, between two strings is the minimal number of insertions, deletions and replacements to make two strings equal [30].

- Jaro similarity between two string is shown in equation 9, where m is the number of matching characters, t is the number of transposition, $|s_1|$ and $|s_2|$ are string length.

$$Jaro(s_1, s_2) = \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}) \tag{9}$$

- Path-similarity is based on WordNet synset, as groups of synonymous words. The similarity between two terms is the shortest path that connects the senses in the "is-a" taxonomy.

- Weighted Overlap is based on NASARI vector constructed using WordNet synsets and Wikipedia pages [6]. Given a pair of words w_i and w_j the algorithm checks if they are synonym, returning maximum similarity score if true. If they are not synonym, it gets their respective NASARI vector and calculates weighted overlap synonym. Weighted Overlap sorts the elements of each vector and harmonically weights the overlap between two vectors. Equation 10 defines weighted overlap for two vectors v_i and v_j, where O is the set of overlapping dimensions between the vectors and r_q^j is the rank of dimension q in the vector v_j.

$$WO(v_i, v_j) = \frac{\sum_{q \in O}(r_q^i + r_q^j)^{-1}}{\sum_{k=1}^{|O|}(2k)^{-1}} \tag{10}$$

5 Experimental Evaluation

Our goal is to analyze the quality of mappings generated in our approach for KG alignment. In the developed experiments, we used datasets from the OAEI (Ontology Alignment Evaluation Initiative) – Knowledge Graph Track released on 2019[7]. Datasets were created running DBpedia extraction framework on Wikis from Fandom Wiki hosting platform [17]. Each instance entity is created from an wiki page and one triple is created for each entry in an infobox [18]. Table 1 describes statistics from the datasets used in our experimental evaluation. The source column have the name of the KG and the acronym used in this work.

Table 1: Statistics of Knowledge Graphs [17]

Source	Hub	Topic	# Instances	# Properties	# Classes
Star Wars Wiki (SWW)	Movies	Entertainment	145,033	700	269
The Old Republic Wiki (TOR)	Games	Gaming	4,180	368	101
Star Wars Galaxies Wiki (SWG)	Games	Gaming	9,634	148	67
Marvel Database (MDB)	Comics	Comics	210,996	139	186
Marvel Cinematic Universe Wiki (MCU)	Movies	Entertainment	45,828	325	181
Memory Alpha (MAL)	TV	Entertainment	45,828	325	181
Star Trek Expanded Universe (STX)	TV	Entertainment	13,426	202	283
Memory Beta (MBT)	Books	Entertainment	51,323	423	240

The mappings created by our proposed approach were compared with the gold standard offered by OAEI – Knowledge Graph Track (2019 edition). The schema level maps were created by experts. Instance level maps were extracted using links present in sections with header containing "link" to corresponding page of another wiki (e.g. "External links"), removing all links where the source page linked to more than one page in another wiki and multiple links to the same concepts to ensure injectivity. The gold standard is a partial gold standard, because it does not contain all correct matches. Trivial match is an exact string match of the label and non-trivial match is when string match is not exact [20]. Table 2 describes statistics of the gold standard used in our experiments.

The learning process used 100% of the mappings to training and validation between the datasets Star Wars Wiki and Star Wars Galaxies Wiki; Star Wars Wiki and The Old Republic Wiki; Memory Alpha and Memory Beta; and Memory Alpha and Star Trek Expanded Universe. Mappings between Marvel Cinematic Universe and Marvel Database were isolated to be used as evaluation set. To this end, we applied our solution to them and analyzed the results based on objective metrics.

MinHash Locality-sensitive hashing used 256 permutation, threshold of 0.75 and each entity were converted to a set of trigrams of entity name to hash. Trigram is a sequence of three consecutive character.

[7]http://oaei.ontologymatching.org/2020/knowledgegraph/index.html

Table 2: Gold Standard statistics [20]

Mapping	Class Matches		Property Matches		Instance Matches	
	Total	Non-trivial	Total	Non-trivial	Total	Non-trivial
SWW-SWG	5	2	20	0	1,096	528
SWW-TOR	15	3	56	6	1,358	220
MCU-MDB	2	0	11	0	1,654	726
MAL-MBT	14	10	53	4	9,296	2,140
MAL-STX	13	6	41	3	1,725	274

We used three metrics to evaluate the results: Precision, Recall and F-Measure. These metrics were used by comparing results obtained by our approach and expected results from the gold standard. Precision is defined as the relation between the number of correctly identified mappings and the number of identified mapping (Formula 11).

$$Precision = \frac{\#IdentifedAndCorrectMappings}{\#IdentifiedMappings} \tag{11}$$

Recall is defined as the relation between the number of correctly identified mappings and the number of expected mappings (Formula 12).

$$Recall = \frac{\#IdentifedAndCorrectMappings}{\#CorrectMappings} \tag{12}$$

F-measure is the harmonic mean of precision and recall (Formula 13).

$$F-Measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{13}$$

Table 3 presents the obtained results in terms of precision, recall and f-measure for schema type. In the class type, we achieved precision higher than 0.5 in the datasets "Star Wars Wiki" and "Star Wars The Old Republic Wiki" (SWW-TOR). In the datasets, "Memory Alpha - Memory Beta" (MAL-MBT) and "Memory Alpha and Star Trek Expanded Universe" (MAL-STX), our solution presented recall lower than 0.9. The class matching for "Memory Alpha" had more Non-trivial mapping compared to other data sets, so it leads to lower recall. In the analysis of the mappings regarding the properties, all datasets presented precision near 0.5 and recall higher than 0.9.

Table 4 presents the results for the mappings connecting the instances of the KGs. All datasets presented low precision, but achieved a recall higher than 0.7 except "Marvel Cinematic Universe - Marvel Database" (MCU-MDB) and "Star Wars Wiki and Star Wars Galaxies" (SWW-SWG).

Figures 7 and 8 compare our approach to the baseline matchers offered by OAEI Knowledge Graph Track organization. The baseline matcher used the label for each entity to create a mapping. The baseline matcher matches all resources which share the same *rdfs:label*. The baseline Alt Label additionally uses *skos:altLabel* as predicate. Both baseline matchers used cross product for all resources that have common label.

Table 3: Mapping results for schema-type entities applying our solution

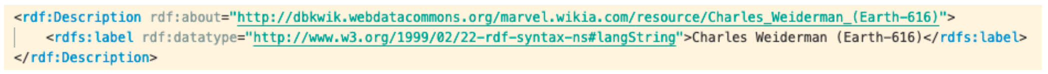
Mapping	**Type**	**Precision**	**Recall**	**F-measure**
SWW-SWG	class	0.385	1.000	0.556
	property	0.435	1.000	0.606
SWW-TOR	class	0.515	0.944	0.667
	property	0.514	0.966	0.671
MCU-MDB	class	0.222	1.000	0.364
	propety	0.611	1.000	0.759
MAL-MBT	class	0.217	0.333	0.263
	property	0.559	0.963	0.707
MAL-STX	class	0.307	0.571	0.400
	property	0.549	0.975	0.703

Table 4: Mapping results for instance-type entities applying our solution

Mapping	**Precision**	**Recall**	**F-measure**
SWW-SWG	0.337	0.494	0.400
SWW-TOR	0.273	0.746	0.400
MCU-MDB	0.357	0.460	0.403
MAL-MBT	0.298	0.739	0.425
MAL-STX	0.228	0.794	0.354

KGs is represented as RDF formed by triples (subject, predicate, object) and data from RDF can be retrieved matching those triples. Baseline matchers query for all triples in the KG that match predicate and *rdfs:label* or *skos:altLabel*. Each entity is a subject in the triple and the object is used to match entities. Figure 6 shows an example of triple that matches predicate and *rdfs:label* extracted from Marvel Database (MDB) [8]. In this example, `http://dbkwik.webdatacommons.org/marvel.wikia.com/resource/Charles_Weiderman_(Earth-616)` is the subject, the predicate is *rdfs:label* and the object is *Charles Weiderman (Earth-616)*.

Our approach and both baselines found all class mappings. Our approach found some false positive classes, so it lowered precision and f-measure. Our approach found all properties mapped and baseline found 36%, but all mapping found by baseline is correct. Overall, our approach exceeded baseline for properties. Figure 7 shows the results obtained by our approach and the baseline matchers for schema-type entities mapping in "Marvel Cinematic



```
<rdf:Description rdf:about="http://dbkwik.webdatacommons.org/marvel.wikia.com/resource/Charles_Weiderman_(Earth-616)">
    <rdfs:label rdf:datatype="http://www.w3.org/1999/02/22-rdf-syntax-ns#langString">Charles Weiderman (Earth-616)</rdfs:label>
</rdf:Description>
```

Figure 6: RDF XML example removed from Marvel Database (MDB)

[8]`http://oaei.webdatacommons.org/tdrs/testdata/persistent/knowledgegraph/v3/suite/marvelcinematicuniverse-marvel/component/target/`

Universe" (MCU) and "Marvel Database" (MDB).

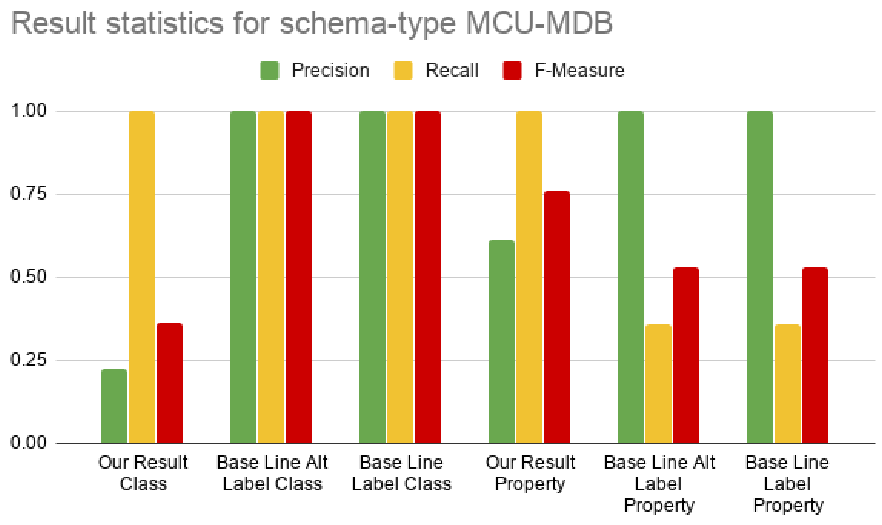


Figure 7: Comparison between our results and competition base line for schema-type entities in "Marvel Cinematic Universe" (MCU) and "Marvel Database" (MDB) mappings.

Figure 8 shows results for instance-type entities in "Marvel Cinematic Universe" (MCU) and "Marvel Database" (MDB). Baseline matcher presents better precision for all cases, but for recall the result is mixed. Our approach had better recall for property, same recall for class and worst for instance.

6 Discussion

This investigation aimed to create mappings between KGs based on rank aggregation methods. Our results by experimenting our approach showed high recall for schema-type, but penalized the precision. For instance level type of entities, our approach presented low precision and acceptable recall.

The main difference between schema-type and instance-type mapping generations is in how candidate entities was created. Our approach used all entities as candidate entities for schema-type, but candidate entities is filtered using locality-sensitive hashing for instance type. For instance-type, it is not possible to use cross-product approach to compare entities, because the number of instance-type entities is very large, so it needs to reduce the quantity of comparisons. For this reason, we were unable to calculate similarity for all possible pair. This difference is more clear for SWW-SWG and MCU-MDB, because both datasets have proportionally more non-trivial mappings than others.

Our approach found false positive mappings which affected the precision for almost all datasets studied. The explored gold standard is based on links created by Wiki community in the page where entities were extracted. It means that the presence of mapping between

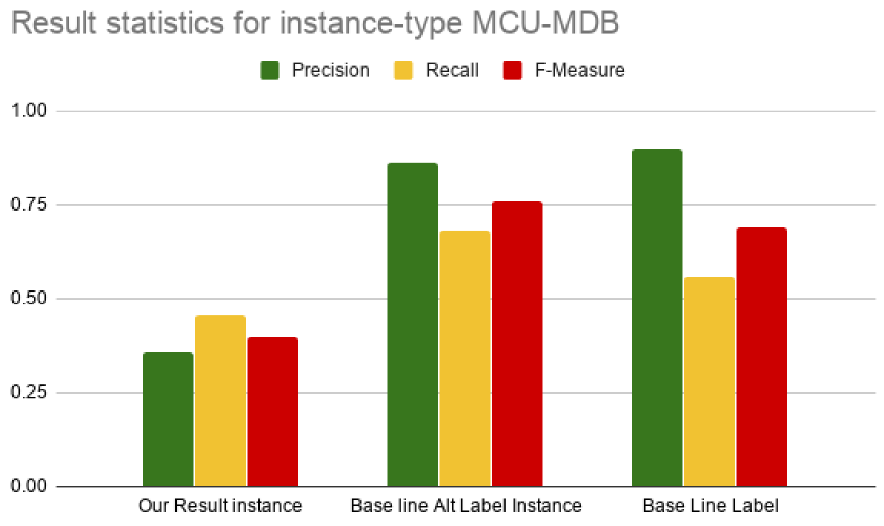


Figure 8: Comparison between our results and competition base line for instance-type entities in “Marvel Cinematic Universe” (MCU) and “Marvel Database” (MDB) mappings.

entities depends on the interest of the community to enrich those pages. For example, “Marvel Database” has the entity identified by `http://dbkwik.webdatacommons.org/marvel.wikia.com/resource/Iron_Man_2:_The_Official_Movie_Storybook` and “Marvel Cinematic Universe” has the entity identified by `http://dbkwik.webdatacommons.org/marvelcinematicuniverse.wikia.com/resource/Iron_Man_2:_The_Official_Movie_Storybook`. Our approach created a mapping between both entities, but it is not contained in the gold standard.

Another case of false positive was caused by different specificities of the entities. In this case, the matcher creates new mappings between an entity and the more general entity, but the correct mapping was for the more specific entity. For example, our technique created a mapping between “Michael Duffy” from “Marvel Cinematic Universe” and “Michael Duffy” from “Marvel Database”. However, the correct answer was between “Michael Duffy” from “Marvel Cinematic Universe” and “Michael Duffy (Earth-616)” from “Marvel Database”.

Our approach found mapping with good recall for most cases in schema-type entities. With property, it exceeded base line recall. Our approach uses learning to rank technique that can be improved with more similarity techniques to aggregate ranking and more dataset without changing code structure.

7 Conclusion

The alignment of KGs remains an open research challenge. In this work, we proposed an approach based on rank aggregation and locality-sensitive hashing to create mappings between distinct KGs. Our approach used the entity URI to extract the set used to explore locality-senstive hashing and similarities. We explored the hashing and four similarity techniques to create independent rankings that were aggregated using learning to rank techniques (in particular, we explored *lambdaMart*). We implemented the proposal and carried out experiments using OAEI competition datasets. Our solution was able to found most of mappings between schema level entities (good recall) although improvements are needed in terms of precision. Future work involves exploring more information from entities to get better results for hashing. We plan to explore other similarity techniques that do not use string as the main component. We also plan to experiment our solution with additional datasets.

Acknowledgements

We thank the São Paulo Research Foundation (FAPESP) (grant #2017/02325-5)[9].

References

[1] Tim Berners-Lee. Uniform resource identifier (uri): Generic syntax. `https://tools.ietf.org/html/rfc3986#section-1.1`, Jan 2005.

[2] R.V. Brickley, Dan; Guha. Rdf schema 1.1. `https://www.w3.org/TR/rdf-schema/`, Feb 2014.

[3] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching*, pages 1–10, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[4] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 89–96, New York, NY, USA, 2005. Association for Computing Machinery.

[5] Chris J.C. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, Microsoft Research, June 2010.

[6] José Camacho-Collados, Mohammad Taher Pilehvar, and Roberto Navigli. NASARI: a novel approach to a semantically-aware representation of items. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 567–577, Denver, Colorado, May–June 2015. Association for Computational Linguistics.

[7] Fei Chang, Guowei Chen, and Songmao Zhang. Fcamap-kg results for oaei 2019. In *Ontology Matching at International Semantic Web Conference, OM@ISWC*, 2019.

[9] The opinions expressed in here are not necessarily shared by the financial support agency.

[8] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In Olivier Chapelle, Yi Chang, and Tie-Yan Liu, editors, *Proceedings of the Learning to Rank Challenge*, volume 14 of *Proceedings of Machine Learning Research*, pages 1–24, Haifa, Israel, 25 Jun 2011. PMLR.

[9] Ricardo Cruz, Kelwin Fernandes, Jaime Cardoso, and Joaquim Costa. Tackling class imbalance with ranking. *IEEE*, pages 2182–2187, 07 2016.

[10] Ronan Cummins and Ted Briscoe. Learning to rank. *Advanced Topics in Natural Language Processing*, 2015.

[11] David; Lanthaler; Markus Cyganiak, Richard; Wood. Rdf 1.1 concepts and abstract syntax. `https://www.w3.org/TR/rdf11-concepts/`, Feb 2014.

[12] Harris Drucker and Corinna Cortes. Boosting decision trees. *Advances in Neural Information Processing Systems*, 8:479–485, 01 1995.

[13] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *12th International Conference on Semantic Systems - SEMANTiCS2016*, 09 2016.

[14] Daniel Faria, Catia Pesquita, Emanuel Santos, Matteo Palmonari, Isabel Cruz, and Francisco Couto. The agreementmakerlight ontology matching system. In *OTM 2013: On the Move to Meaningful Internet Systems: OTM 2013 Conferences pp 527-541*, volume 8185, 09 2013.

[15] Jerome Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29, 11 2000.

[16] Shikhar Gupta. Locality sensitive hashing. `https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134`, Apr 2019.

[17] Sven Hertling and Heiko Paulheim. Knowledge graph track. `http://oaei.ontologymatching.org/2020/knowledgegraph/index.html`.

[18] Sven Hertling and Heiko Paulheim. Dbkwik: A consolidated knowledge graph from thousands of wikis. In *2018 IEEE International Conference on Big Knowledge (ICBK)*, pages 17–24, 11 2018.

[19] Sven Hertling and Heiko Paulheim. Dome results for oaei 2019. In *OM 2019 : Proceedings of the 14th International Workshop on Ontology Matching co-located with the 18th International Semantic Web Conference (ISWC 2019) Auckland, New Zealand, October 26, 2019*, volume 2536, pages 123–130, Aachen, 2019. RWTH.

[20] Sven Hertling and Heiko Paulheim. *The Knowledge Graph Track at OAEI: Gold Standards, Baselines, and the Golden Hammer Bias*, pages 343–359. Springer; 1st ed. 2020 edition, 05 2020.

[21] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs, 2020.

[22] Jaccard. The distribution of the flora of the alpine zone. In *New Phytologist*, volume 11, pages 37–50, 1912.

[23] Julio Cesar dos Reis Juliana M. Destro, Javier A. Vargas and Ricardo da S. Torres. Exploring rank aggregation for cross-lingual ontology alignments. *Proceedings of the 14th International Workshop on Ontology Matching co-located with the 18th International Semantic Web Conference (ISWC 2019)*, 2019.

[24] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[25] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6, 01 2014.

[26] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets.* Cambridge University Press, USA, 2nd edition, 2014.

[27] Bopeng Li, Sougata Chaudhuri, and Ambuj Tewari. Handling class imbalance in link prediction using learning to rank techniques. *ArXiv*, abs/1511.04383, 2016.

[28] Hang LI. A short introduction to learning to rank. *IEICE Transactions on Information and Systems*, E94.D(10):1854–1862, 2011.

[29] Claudia Müller-Birn, Benjamin Karran, Janette Lehmann, and Markus Luczak-Roesch. Peer-production system or collaborative ontology engineering effort: What is wikidata? In *OpenSym 2015*, 08 2015.

[30] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[31] Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman. *Mining Massive Datasets.* Cambridge University Press, 2014.

[32] Daniel Ringler and Heiko Paulheim. One knowledge graph to rule them all? analyzing the differences between dbpedia, yago, wikidata & co. In Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm, editors, *KI 2017: Advances in Artificial Intelligence*, pages 366–372, Cham, 2017. Springer International Publishing.

[33] Daniel Ringler and Heiko Paulheim. One knowledge graph to rule them all? analyzing the differences between dbpedia, yago, wikidata co. In *978-3-319-67189-5*, pages 366–372, 09 2017.

[34] Amit Singhal. Introducing the knowledge graph: things, not strings. `https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html`, 2014.

[35] Fabian Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, 01 2007.

[36] Yi Hsuan Tsai and Ming Hsuan Yang. Locality preserving hashing. In *2014 IEEE International Conference on Image Processing, ICIP 2014*, 2014 IEEE International Conference on Image Processing, ICIP 2014, pages 2988–2992, United States, January 2014. Institute of Electrical and Electronics Engineers Inc.