AnyGraphMatcher Submission to the OAEI Knowledge Graph Challenge 2019*

Alexander Lütke[1]

University of Mannheim, Germany

Abstract. Matching objects between two different data bases typically relies on syntactic similarity measurements and the exhausting of ontology restrictions. As opposed to them, AnyGraphMatcher (AGM) introduces an additional source of information for semantically matching data – i.e. the creation of word embeddings. AGM's key idea wraps around a stable marriage for determining best matching data objects between two data bases. Results on the OAEI knowledge graph track however indicate the need for a more advanced blocking technique. Results show that word embeddinngs are to be seen a supportive feature for mapping rather than a key source of information.

Keywords: Ontology matching · Word embeddings · Semi-supervised machine learning.

1 Presentation of the system

1.1 State, purpose, general statement

In recent years, data has developed into a differentiator for business success. But organizations gathered data typically comes from various sources, which are mutually heterogeneous and inconsistent. Identity resolution is required to locate and integrate common pieces of information between these data sources. More precisely, data elements between those data bases have to be compared to each other and a decision on whether they describe the same real world concept must be made. Most prevalent techniques resolve around the comparison of syntactic elements, like titles, labels or descriptions. However, those techniques fail to preserve the actual semantic meaning of data objects. Consider for example the word Berlin, either describing Germany's capital or a cargo ship. Just from the title, the semantic meaning of the word "Berlin" cannot be determined. Recent breakthrough in linguistics research on the latent representation of words offers a promising opportunity [3]. The main notion wraps around the distributional hypothesis by Harris, stating that the meaning of a word is defined by its context. First, the hypothesis referred to linguistics only. But the adaptation in Paulheim's RDF2Vec approach [5] showcased, that the same concept is applicable to (semi-)structured databases. ALOD2VEC [4] and DOME [1] are two

systems, which already used the idea of word embeddings for data integration in the 2018 OAEI challenge. Compared to them, AnyGraphMatcher (AGM) is a novel concept specifically for identity resolution, which extends RDF2Vec with the use of semi-supervised machine learning and stable marriage.

1.2 Specific techniques used

The task of identity resolution is perceived as a binomial statement whether two given data elements describe the same real-world object. The final goal is such a binomial prediction for the entire Cartesian product of all elements of two databases A and B. Essentially, AnyGraphMatcher employs a five step process to get there:

1. Blocking
2. Graph walk
3. Word embedding model
4. Semi-supervised machine learning
5. Stable marriage

The Cartesian product can be extremely large depending on the size of the input datasets, representing a burden to runtime performance. However, most of the object pairs from the Cartesian product will not be matching, particularly if the input datasets are considered free of duplicates. So **blocking** is required. Instead of performing the following, computationally expensive predictions on the whole Cartesian product, a number of candidate-pairs is chosen based on an efficient similarity computation first. All pairs except the candidate-pairs are directly predicted non-matching. For the efficient candidate selection, a levenshtein automata is utilized, which measures *syntactic* distances. More precisely, edit distances up to two edits per word in a string are calculated. Note one major limitation in this concept: An assumption is met that actually corresponding data objects have similar labels. This might increase precision, but decrease recall.

Afterwards, a **graph walk** is employed, which iterates through the set of vertices and edges of an ontology graph. While walking through the graph, visited paths are written down, so that a corpus file is created in the end. The more detailed, recursive procedure looks as follows: For each vertice, an outgoing edge is selected. The edge is traced to its endpoint and the selection-procedure is started from there on again. At each vertice visited, such a selection is triggered n times. Furthermore, after excelling a distance of k steps from the vertice, where the path started, the procedure is terminated. Due to runtime limitations, for the OAEI submission, n equalling the number of outgoing edges of the current vertice and $k = 1$ was chosen. Basically, this boils down to a simple NT-file representation of an ontology. Further improvements could be derived from experiments with larger k values.

The graph walk has generated a text corpus, which can be passed to a **word embedding model** to form a latent representation of each word occurring in the

corpus. As the utilized SkipGram neural model is quite prevalent today, further details on the concept remains to the respective works in the literature. Since the SkipGram model is however highly configurable, most parameters have been adopted from the general recommendation of the inventors (e.g. hidden layer size = 100) Only the number of epochs has been modified. Due to runtime limitations, those depend on the size of the generated corpus, but halts between 10 and 250. After running the SkipGram model on the corpus, each resource in the input ontology can be assigned a 100-dimensional vector.

The final goal is a prediction whether two candidate pairs correspond to each other. This is achieved by **semi-supervised machine learning**. Supervision requires for a gold standard, which cannot be presupposed for said mapping tasks. That is why AnyGraphMatcher employs an automatic gold standard set generation. For this purpose, the candidate-pairs from the blocking are considered once again. Special attention is paid to the similarity of data object's labels. For the efficient calculation, the library "Apache Lucene" is used, which measures similarity in a value range of up to 2.5 for identical strings and 0.0 for very distant strings.[1] Very similar candidate-pairs with a similarity-score of 2.0 or above are assumed matches. All further candidate-pairs, in which one of the matching data objects occurs, are assumed non-matches. In the end, this gold standard captures most apparent matches, but differentiates them by the inclusion of matches (i.e. positives) and non-matches (i.e. negatives).

With the gold standard in place, a binary machine learning classifier – here an XGBoost – can work properly. Besides, the classifier takes more information than just the syntactic similarity into account. I.e. latent information is derived from the SkipGram model and passed to the classifier. Pairwise cosine similarity, Euclidean distance and SkipGram context probability (i.e. the probability that resource x appears in the context of resource y) are calculated. Other than expected, the binary classification is not meant to be the final prediction step however. Rather, it is implemented as another, more enhanced way of blocking. This is done by up-sampling the matching pairs during training until there are 1.5 times as many matches as non-matches. This procedure makes sure, that only *very likely* non-matches are classified negatives and excluded from further processing.

The final prediction is achieved by **stable marriage** under the assumption of 1:1 cardinality mappings. Here, each data object is considered in isolation first. All remaining candidate-pairs, in which a given data object appears, are extracted. For all found candidate-pairs, an overall similarity score is calculated. That score includes cosine similarity, Euclidean distance, SkipGram context probability and levenshtein distance. For each of these measures, the relative similarity in comparison with other candidate-pairs is computed. This is quantified by the position p in the following formula:

$$sim_{relative} = 2^{-(p-1)} \tag{1}$$

[1] The exact calculation is not meant to be explained here. For further details, refer to https://lucene.apache.org/core/3_5_0/scoring.html.

The following tables 1.2, 1.2 and 1.2 illustrate the idea once more. The focused data object in these tables is called A. A might match to B, C and D. Table 1.2 shows various thought similarity values between the candidate objects and A. Stable marriage goes on as follows to determine which one of the candidates is the best matching one: Table 1.2 indicates the order, how well a candidate-pair matches compared to other candidate pairs. Note that each of the similarity measure is still considered in isolation here. Table 1.2 then translates the ordering into values computed by the equation above. The final score in table 1.2 is calculated by summing the translated values for each of the candidate pairs. The one pair with the highest total score is assumed to match best. As a secondary criterion for further discrimination, the score derived from Levenshtein distance is taken. The way the final score is calculated is to be seen as an optimisable characteristic of AGM.

Candidate-pair	**Cos. sim**	**Eucl. dist.**	**Lev. dist.**	**P(Context)**
A – B	0.5	1.5	4	0.1
A – C	0.8	1.0	8	0.15
A – D	0.7	1.4	5	0.2

Table 1. Similarity measures of candidate pairs

Candidate-pair	**Relative Position**			
	Cos. sim	**Euclid. dist**	**Lev. dist.**	**P(Context)**
A – B	3	3	1	3
A – C	1	1	3	2
A – D	2	2	2	1

Table 2. Ordering candidate pairs based on their relative similarity

Candidate-pair	**Preliminary score**				**Total score**
	Cos. sim	**Euclid. dist**	**Lev. dist.**	**P(Context)**	
A – B	0.25	0.25	1	0.25	1.75
A – C	1	1	0.25	0.5	**2.75**
A – D	0.5	0.5	0.5	1	2.5

Table 3. Ordering candidate pairs based on a final score calculation

1.3 Summary of the system's limitations

Despite AnyGraphMatcher's exploitation of a wide range of characteristics of data objects, it suffers from two major limitations:

First, strong confidence is set to syntactic similarity. Basically two assumption lead to this conclusion: (1) data objects, which are syntactically similar, do match (see gold standard generation) and (2) actually matching data objects have similar labels (see blocking).

Second limitation is the recall-bias in the entire pipeline. Note that the only three steps in place to predict candidate-pairs negative are (1) blocking, (2) semi-supervised machine learning and (3) stable marriage. Blocking (1) and semi-supervised machine learning (2) themselves are recall-biased. So they prefer to predict (syntactically) similar samples as positives rather than negatives. Stable marriage can only predict negatives, if a data object has already been identified a match with another data object. So all in all, there is no strict exclusion of negatives from the set of candidate-pairs. This might raise precision, but reduce recall.

In sum, AGM can basically exploit semantics, if and only if the underlying data sets consistently follow the syntactic similarity assumption from above.

1.4 Adaptations made for the evaluation

For the OAEI submission, the melt framework provided by the University of Mannheim has been used [2]. Melt handles most of the regulations required for submitting matcher systems to the OAEI challenge. Since melt is originally written in Java, while AGM is mainly developed in Python, melt is used as a wrapper service, that calls the AGM pipeline by starting a new Python-process. Furthermore, the blocking process has been adapted to run more efficiently on the larger of the data sets in the OAEI knowledge graph track. A very strict blocking is applied, that initially excludes a lot candidate matches. Whether this technique harms recall is to be clarified in the results section.

1.5 Link to the system and parameters file

The implementation of AGM can be found on Github using the link https://github.com/XLexxaX/AnyGraphMatcher/tree/SUBMISSION.

2 Results

The following paragraphs shortly outline the results of AGM compared to the baseline figures. It therefore refers to the full result table available online on http://oaei.ontologymatching.org/2019/results/knowledgegraph/index.html. For the reason of comprehensibility, table 4 lists a more compact overview of the knowledge graph track.

System	Prec.	F-m.	Rec.
AGM	0.48 (0.48)	0.25 (0.25)	0.17 (0.17)
AML	0.72 (0.90)	0.70 (0.88)	0.69 (0.86)
baselineAltLabel	0.89 (0.89)	0.84 (0.84)	0.80 (0.80)
baselineLabel	0.95 (0.95)	0.81 (0.81)	0.71 (0.71)
DOME	0.74 (0.92)	0.70 (0.88)	0.67 (0.84)
FCAMap-KG	0.91 (0.91)	0.85 (0.85)	0.79 (0.79)
LogMapKG	0.40 (0.40)	0.54 (0.54)	0.84 (0.84)
LogMapLt	0.73 (0.91)	0.66 (0.83)	0.61 (0.76)
Wiktionary	0.91 (0.91)	0.80 (0.80)	0.71 (0.71)

Table 4. Comprehensive overview of the OAEI knowledge graph track results

2.1 Marvel Cinematic Universe Wiki ∼ *MarvelDatabase*

With an overall F-score of 11%, AGM fails to output proper mappings on the first of the five mapping tasks. Taking a closer look at the five steps in the AGM pipeline, the exclusion of many candidate-mappings during blocking stands out.

2.2 Memory Alpha ∼ *MemoryBeta*

The mapping of Memory Alpha to Memory Beta yielded slightly better results with an F-score of 32%. But still, the purely syntax-based baseline oupterforms AGM by approximately 50%. Notable is however, that this time, precision (47%) is significantly better than recall (24%).

2.3 Memory Alpha ∼ *StarTrekExpandedUniverse*

The observations from Memory Alpha and Memory Beta continue throughout the remaining three mapping tasks. All in all, an F-score of 30% was achieved when mapping Memory Alpha to Star Trek Expanded Universe. The baseline of 91% F-score is missed.

2.4 Star Wars Wiki ∼ *StarWarsGalaxiesWiki*

For the Star wars wiki mapping, again 30% F-score is achieved. The baseline F-score of 67% is out of reach. Note however the even larger gap of 52% between AGM's precision and recall this time.

2.5 Star Wars Wiki ∼ *TheOldRepublicWiki*

Repeatedly, a 52% gap between recall and precision is conspicious. The F-score of 20% does not meet the pretension of the baseline by far.

3 General comments

3.1 Comments on the results

In sum, results of AGM on the knowledge graph track are relatively weak compared to the baseline figures. Mainly recall is lacking behind the competition's results. This can be traced back to the strict blocking technique. However, precision also lacks behind the baseline figures. Recap, that a levenshtein automata has been used, which can only measure edit distances of up to 2 edits per word. In case two long texts are compared, this restriction leads to imprecise measuring. So the levenshtein automata as implemented in AGM is rather an approximation of syntactic similarity. Nevertheless, a static threshold has been used for blocking (see section 1.2), such that in the end precision suffers as well.

3.2 Discussions on the way to improve the proposed system

In order to compensate for the weak results, another way has to be found block based on a weaker threshold, while ensuring runtime efficiency of the AGM pipeline. One idea is to loosen the current threshold and introduce a second blocking step, that blocks based on exact edit distances for all candidates found by the levensthein automata.

4 Conclusion

AGM follows a novel approach to data mappings by utilizing the idea of word embeddings. It implements a five step process including blocking, a graph walk, embedding creation, semi-supervised machine learning and stable marriage. By combining different similarity measures derived from syntax and word embeddings, AGM aims to yield semantically correct mappings. However results show a relatively poor performance compared to the purely syntax based baseline figures. A strict and imprecise blocking technique has been identified a root cause. Though the results cannot achieve the baseline figures, they provide a valuable outcome for AGM's approach in general: The stable marriage depends a lot on the upstream steps and suffers from error-propagation. This implies that features derived from embeddings cannot be solely used for mapping. Embeddings are to be seen an approximation of concept's semantic meaning, such that they can additionally support in distinguishing them. In order to compensate for this observation in the future, a more advanced blocking technique is required.

References

1. DOME results for OAEI 2018. In *OM@ISWC*, volume 2288 of *CEUR Workshop Proceedings*, pages 144–151, Karlsruhe, 2018. CEUR-WS.org.
2. Sven Hertling, Jan Portisch, and Heiko Paulheim. MELT - Matching EvaLuation Toolkit. In *Semantics 2019 SEM2019 Proceedings*, Karlsruhe, 2019, to appear.

3. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.
4. Jan Portisch and Heiko Paulheim. Alod2vec matcher. In *OM@ISWC*, volume 2288 of *CEUR Workshop Proceedings*, pages 132–137. CEUR-WS.org, 2018.
5. Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *The Semantic Web - ISWC 2016 : 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, volume 9981, pages 498–514, Cham, 2016. Springer International Publishing.