ATBox Results for OAEI 2020

Sven Hertling[0000−0003−0333−5888] and Heiko Paulheim[0000−0003−4386−8195]

Data and Web Science Group, University of Mannheim, Germany
{sven,heiko}@informatik.uni-mannheim.de

Abstract. ATBox matcher is a scalable system for instance (Abox) and schema (Tbox) matching. It uses two pipelines for generating candidates for the schema and instance matching, and utilizes the schema matches to further improve the instance correspondences. Using a string blocking method, ATBox is able to align large ontologies and can run on OAEI tracks like largebio and knowledge graph. The results look promising, but further features for better finding correct instance matches can be developed.

Keywords: Ontology Matching · Knowledge Graph

1 Presentation of the system

Nearly all systems submitted to the Ontology alignment Evaluation Initiative (OAEI) are able to align ontologies, schemas, or Tboxes, as they are called in description logics (DL). On the other hand, there are more and more instance tracks like spimbench, link discovery, geolink cruise, and knowledge graph, matching instances, or Aboxes, becomes equally important. The matcher presented in this paper, called *ATBox*, focuses on both the Abox and Tbox.

Especially the knowledge graph track needs scalable systems which can deal with hundred of thousands of instances [4]. Thus, the basis of this matcher is a blocking approach, which focuses on high recall. Its result is succesively fine tuned to increase the precision. Given this design, ATBox is also able to match large knowledge graphs like DBpedia [1] or YAGO [6].

1.1 State, purpose, general statement

The overall matching strategy of ATBox is shown in figure 1. The Tbox and Abox have different processing pipelines but the correspondences are combined in the end to get the final alignment.

Tbox matching is applied for all classes and properties (`owl:ObjectProperty`, `owl:DatatypeProperty`, and `rdf:Property`). They are retrieved by the jena[1] methods *OntModel.listClasses()* and *OntModel.listAllOntProperties()*.

[1] https://jena.apache.org

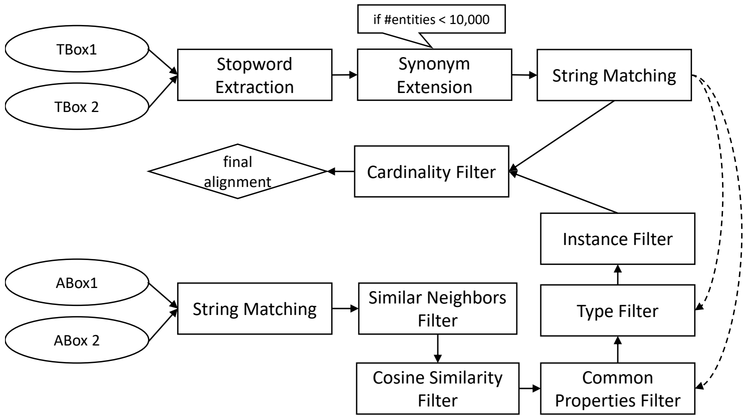

Fig. 1. Overview of the ATBox matcher strategy.

The Tbox matching (classes and properties) starts with the stopword extraction. In some cases the labels and/or fragments (which we define as the part after the last hashtag symbol # or slash /) contains tokens which appears very often like `class`, `infobox` etc. If such tokens appears in more than 20 % of all classes/properties (considered separately), then it is extracted as a corpus specific stop word. In case there are many such stop words, they are restricted to the five most occurring ones.

The synonyms (used during string matching) are extracted from the English Wiktionary to cover many different domains. The extraction is done with DBnary [8], a dataset containing Wiktionary as RDF. The extraction process starts with all resources of type dbnary:Page [2] within the English domain [3]. Then we follow the `describes` relation and extract all resources connected with property `synonym`. Furthermore we follow the relation `sense` to also find all the given senses and their synonyms. The lemmas are extracted directly from the URI.

Table 1. String processing steps in ATBox matcher for schema matches.

Processsing	Confidence	Levenshtein
equality	1.0	no
normalize	0.9	no
normalizeParentheses	0.8	no
defaultStopwords	0.7	no
corpusStopwords	0.6	yes
synonyms	0.5	no

[2] http://kaiko.getalp.org/dbnary#Page
[3] http://kaiko.getalp.org/dbnary/eng/

The string matching contains multiple different steps which are shown in table 1. All processing applies to `rdfs:label` and in case it is missing to the URI fragment. If the extracted text is exactly the same, the generated correspondence has a confidence of 1.0. During the normalization process, a word written in camel case[4] is separated with whitespace (e.g. `hasAge` to `has Age`) and afterwards lowercased. In case some UTF-8 characters are not normalized, we apply a normalization step for them (e.g. an accented character can be encoded in multiple different ways in UTF-8). All possible punctuations are furthermore removed and multiple whitespaces are combined into one. In case the normalized text matches, a confidence of 0.9 is assigned. In the `normalizeParentheses` step, all text within parentheses is removed. If the remaining normalized text (same as in `normalize` step) is equal, it assigns a confidence of 0.8. The reason behind is that many articles in KGs define concepts with same names to have the discriminating term in parentheses e.g. “Harry Potter (character)” and “Harry Potter (film series)”. `DefaultStopwords` removes a given set of stopwords while keeping all other processing steps as before (confidence is 0.7). In the last processing step, the corpus specific stopwords, extracted before, are also removed and additionally allow a levenshtein distance[7] of 1 (but only in case the text is longer than 6 characters). In case it matches a correspondence with confidence of 0.6 is generated. If the amount of concepts are less than 10,000 for source and target, then a synonym step is added with a confidence of 0.5. In this step, the extracted synonyms are used to replace (possibly multiple) tokens with all available synonyms.

All string processing steps are executed in order starting with the highest confidence. If a match is found the remaining steps are also executed to find possible other candidates. As an example, a correspondence like <Harry_Potter,harry_potter, =, 0.9> is already found, then the processing continues and also add <Harry_Potter,Harry_Potter(Book), =, 0.8> to the resulting alignment.

The instance matching (Abox - shown in the lower part of the figure 1) starts directly with the string matching component. It reuses the processing steps described in the previous section without the corpus dependent stopword removal and synonym replacement. The applied steps are shown in table 2. The first four steps applies to the `rdfs:label` and if it is missing to the fragment of the URI. The confidence is decreasing with a step size of 0.1 starting with 1.0. In the second part, the additional properties `skos:prefLabel` and `skos:altLabel` are taken into account. If they match, the confidence is set to maximally 0.6 depending in which preprocessing step the match occurs. Once again, we allow matches which a lower confidence, even when a correspondence with a higher confidence is found. This increases the recall because it might be the case that the matched entity with a high confidence is not the best available match.

The string processing step generated an alignment with a high recall. All following steps try to increase the precision by generating additional confidences for each correspondence. This helps at the end of the processing pipeline to enforce a one to one alignment and selecting the right correspondence in

[4] https://en.wikipedia.org/wiki/Camel_case

Table 2. Processing steps for generating instance matches.

Processsing	Confidence	Property
equality	1.0	rdfs:label (or fragment)
normalize	0.9	rdfs:label (or fragment)
normalizeParentheses	0.8	rdfs:label (or fragment)
defaultStopwords	0.7	rdfs:label (or fragment)
equality	0.6	+ skos:preflabel, skos:altLabel
normalize	0.5	+ skos:preflabel, skos:altLabel
normalizeParentheses	0.4	+ skos:preflabel, skos:altLabel
defaultStopwords	0.3	+ skos:preflabel, skos:altLabel

case there are multiple target entities for one source entity (or the other way around). Thus the following filters only add additional confidences (with the `addAdditionalConfidence` function of YAAA [5]) and do not yet remove any correspondences:

- Similar Neighbors Filter
- Cosine Similarity Filter
- Common Properties Filter
- Type Filter

All these filters are explained in the following. The similar neighbors filter uses the instance alignment (generated by the previous string processing step) to count for each instance correspondence how many resources or literals are shared between the two instances. Figure 2 shows an example where two neighbors are detected for correspondence <one:Harry_Potter, two:Harry_Potter> because the literal "blue" and the resource "Gryffindor" is shared. Note that the properties are not taken into account (which is done later by the common properties filter). Thus we do not need a mapping of property "eyeColor" to "eye". We further exclude the properties `rdfs:label` and `skos:altLabel` and all properties which have the same literal as those. This will not count the literals which just repeats the name of the resource with a different (maybe not matched) property like "name". Two literals are the same when their lowercased lexical value is equal. The additional confidence is the absolute amount of neighbors.

The cosine similarity filter compares text which is extracted from instances. It is generated by iterating over all literals and checking if the datatype of it is `xsd:string`,`rdf:langString` or if the literal has a language tag. All lexical representations of such literals are concatenated to generate a textual representation. These representations are then compared with a cosine similarity which is added to the correspondence.

The common properties filter checks for each instance correspondence the number of shared properties. This heavily relies on already matched schema because all properties with the same URI are excluded beforehand. Thus we only check if the instances share some matched properties regardless of their objects. The number of overlap is then added to the correspondence.

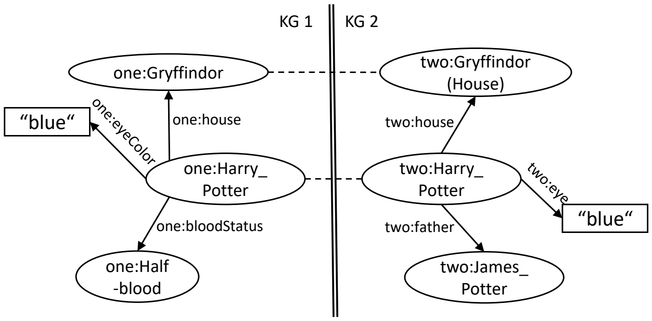

Fig. 2. The similar neighbors filter would assign two neighbors for the correspondence <one:Harry_Potter, two:Harry_Potter> because of literal "blue" and the already matched entites one:Gryffindor and two:Gryffindor(House).

The type filter is similar to the neighbors filter but only checks if the types (retrieved by `rdf:type`) actually overlap. This again requires already matched classes. The absolute overlap is added as an additional confidence.

The final step during instance matching is to actually filter these correspondences and create a one to one alignment. This instance filter sorts the correspondences by confidence (which is initially set by the string matching) and iterating over it. If a source or target resource is already matched, then it continues with the next correspondence. In all other cases it checks if there is a correspondence in the whole instance alignment which should be used instead. The criteria for being better is fixed to have greater values in two additional confidences.

As a last step, all correspondences are combined and a final cardinality filter ensures a one to one alignment by comparing the confidence scores.

1.2 Specific techniques used

We used the following matching components of MELT [5]:

- ScalableStringProcessingMatcher
- StopwordExtraction
- SimilarNeighborsFilter
- CommonPropertiesFilter
- CosineSimilarityConfidenceMatcher
- SimilarTypeFilter
- NaiveDescendingExtractor

1.3 Adaptations made for the evaluation

ATBox matcher is also available as a SEALS package. Due to clashes of dependecies of SEALS and ATBox, we decided to use the external SEALS packaging mechanism of the MELT framework[5]. It generates an intermediate matcher which executes an external process which runs in its own java virtual machine (JVM). Thus different versions of dependencies are not a problem.

1.4 Link to the system and parameters file

ATBox matcher can be downloaded from
https://www.dropbox.com/s/q57rzoec9zeumi2/ATBox.zip?dl=0.

2 Results

This section discusses the results of ATBox for each track of OAEI 2020 where the matcher is able to produce results. The following tracks are included: anatomy, conference, largebio, phenotype, and knowledge graph track.

Specific matching strategies and interfaces for the interactive and complex track are currently not implemented and are thus not described. Due to no multi language support, the multifarm track is also excluded.

2.1 Anatomy

ATBox could achive a slightly higher F-measure than the baseline (0.799 vs 0.766). Even though a synonym step is included in the matcher, the recall is only at 0.671 but therefor a high precision of 0.987 could be achieved (third best value).

Some examples were the matcher could find some non-trivial matches are:

- <cranium, Skull, =, 0.5>
- <lienal vein, Splenic_Vein, =, 0.5>
- <inner ear, Internal_Ear, =, 0.5>
- <celiac artery, Coeliac_Artery, =, 0.6>
- <grey matter, Gray_Matter, =, 0.6>

The first three have a confidence of 0.5 and thus the matches are mainly generated by synonym replacements. The last two contain different spellings like “grey” and “gray”. They are matched because the levenshtein distance is one between the two strings.

Some examples where the synonym step yields wrong results are:

- <naris, Nostril, =, 0.5>
- <upper arm, Biceps, =, 0.5>

This shows that not only true positives are generated and it is also the reason why the correspondence has a low confidence.

2.2 Conference

In the conference track ATBox matcher (0.56) is a bit better in terms of F-Measure than the baselines edna (0.54) and StringEquiv (0.52) when using the ra2-M3 evaluation. It covers the class and property alignments (M3) and uses the ra2 reference alignment which is a transitive closure of the original reference alignment ra1[9]. Analyzing the precision/recall triangular graph which is based on the same evaluation dataset is can easily be seen that ATBox matcher has the best tradeoff between recall and precision. The reason is mainly the higher recall and the lower precision which is not easily avoidable. The schema matching capabilities of ATBox are rather limited and thus only the synonym expansion helps a lot. The ontology specific stopwords do not help here because they do not exist in the given dataset. Some examples where the synonym step help: <Trip, Excursion>, <Participant, Attendee>, <Place, Location>, and <SubjectArea, Topic>. The levenshtein distance helps finding <Sponsor, Sponzor> and <Organization, Organisation>. Furthermore ATBox is one of the seven matching systems which returns a wide variation of confidence values.

2.3 Largebio

ATBox matcher is one of six systems which are able to run on all six test cases and return meaningful alignments. It was consequently the second fastest system after LogMapLt. The results are very good in terms of precision but the recall is to low to compete with the other participants. Only in the FMA-SNOMED small fragments test cases the presented matcher could perform better than Wiktionary and LogMapLt.

2.4 Phenotype

In this track the presented matcher only returns 759 correspondences for the first task HP-MP and 1,318 correspondences for the second task DOID-ORDO. The evaluation result thus contains a low recall of 0.298 respectively 0.333. Together with a high precision, a F-measure of 0.457 and 0.498 can be achieved. This is probably due to the missing background knowledge because LogMapBio uses BioPortal, LogMap uses spelling variants of SPECIALIST lexicon, and AML uses three sources (Uberon, DOID, and MeSH). All these systems achieve a higher recall than ATBox. Nevertheless in task HP-MP we could rank higher than ALOD2Vec and Wiktionary.

2.5 Biodiv

In the Biodiv track ATBox could only return results in FLOPO-PTO test case. Once again the F-measure of 0.714 is much better than those of Wiktionary and ALOD2Vec but less than all LogMap variants and AML.

2.6 Knowledge Graph

ATBox could score int the overall evaluation(which contains classes, properties, and instances) the second highest F-measure score of 0.85 together with AML. Only ALOD2Vec and Wiktionary scores 0.01 better. When matching only classes, the presented matcher is the second best system after AML and for properties it is the best matcher. The instance matching pipeline is helpful for finding the correct correspondences but with 0.84 it is a bit below AML (0.85), ALOD2Vec (0.87), and Wiktionary (0.87).

3 General comments

3.1 Discussions on the way to improve the proposed system

We would like to increase the number of feature generators. For example, all texts connected to an instance could be compared not only with cosine similarity but also with a BERT classifier[2]. Another feature would be to compare images associated with the instances to further distinguish true positive from false positive correspondences.

Furthermore the schema matches could be improved with the help of all instance correspondences as already shown in DOME matcher [3].

4 Conclusions

In this paper, we have analyzed the results of ATBox matcher in OAEI 2020. It shows that the system is very scalable and can generate class, property and instance alignments. It usually has a high precision but on some tracks like Largebio, Phenotype, and Biodiv the recall can be increased by utilizing external knowledge despite the already used synonym lexicon from Wiktionary.

Most of the used matching components are furthermore included in the MELT framework[5] to allow other system developers to reuse them.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: The semantic web, pp. 722–735. Springer (2007)
2. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
3. Hertling, S., Paulheim, H.: Dome results for oaei 2019. OM@ ISWC **2536**, 123–130 (2019)
4. Hertling, S., Paulheim, H.: The knowledge graph track at oaei - gold standards, baselines, and the golden hammer bias. In: The Semantic Web: ESWC 2020. pp. 343–359 (2020)
5. Hertling, S., Portisch, J., Paulheim, H.: Melt - matching evaluation toolkit. In: SEMANTICS. Karlsruhe. (2019)
6. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: Yago2: A spatially and temporally enhanced knowledge base from wikipedia. Artificial Intelligence **194**, 28–61 (2013)
7. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics doklady. vol. 10, pp. 707–710 (1966)
8. Sérasset, G.: Dbnary: Wiktionary as a lemon-based multilingual lexical resource in rdf. Semantic Web **6**(4), 355–361 (2015)
9. Zamazal, O., Svátek, V.: The ten-year ontofarm and its fertilization within the onto-sphere. Journal of Web Semantics **43**, 46–53 (2017)