ATBox Results for OAEI 2021

Sven Hertling and Heiko Paulheim

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 de-
scription 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 [3]. Thus, the basis of this matcher is
a blocking approach, which focuses on high recall. Its result is successively 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:ObjectProp-
erty, owl:DatatypeProperty, and rdf:Property). They are retrieved by the
jena methods OntModel.listClasses() and OntModel.listAllOntProperties().
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. |
| --- | --- | --- |
| Processing | 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](http://kaiko.getalp.org/dbnary#Page)

3 [http://kaiko.getalp.org/dbnary/eng/](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
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 \texttt{addAdditionalConfidence} function of YAAA \cite{4}) 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 \texttt{<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 \texttt{rdfs:label} and \texttt{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 \texttt{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.
The similar neighbors filter would assign two neighbors for the correspondence \(<\text{one:Harry\_Potter}, \text{two:Harry\_Potter}>\) because of literal “blue” and the already matched entities \(<\text{one:Gryffindor, two:Gryffindor(House)}\>.

The type filter is similar to the neighbors filter but only checks if the types (retrieved by \texttt{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 [4]:

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

1.3 Adaptations made for the evaluation

ATBox matcher is also available as a docker based matcher which runs a HTTP endpoint. The matcher is packaged with the MELT framework[4]. It will generate a docker image which also contains the code for running a small server.
1.4 Link to the system and parameters file

ATBox matcher can be downloaded from https://www.dropbox.com/s/l344aawh0mw6rjm/atmatcher-1.0-web-latest.tar.gz?dl=0.

2 Results

This section discusses the results of ATBox for each track of OAEI 2021 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

In comparison to last years participation, the F-Measure slightly decreased from 0.799 to 0.794 but still beats the baseline by a small margin. The matcher is rather precision oriented and achieves the third highest value after the string baseline, LSMatch, and ALIN. Recall should be optimized further than just using synonyms and an alignment repair step can be introduced to make a coherent alignment (which is not the case yet for anatomy).

2.2 Conference

In the conference track, ATBox matcher increased the F-Measure from 0.57 to 0.59 using the rar2-M3 evaluation setup (which is a violation free version of the entailed reference alignment for classes and properties). This is the third highest value after AML, LogMap, and GMap. Again the recall (with 0.51) is lower than precision (with 0.69).

2.3 Largebio

Results not yet published.

2.4 Phenotype

Results not yet published.

2.5 Biodiv

In the Biodiv track ATBox could not return any results for the three test cases. Last year ATBox was actually able to return an alignment for FLOPO-PTO test case. We will investigate why this is not the case for OAEI 2021.
2.6 Common Knowledge Graphs

This is a new track which was introduced in OAEI 2021. The task is to align classes between NELL and DBpedia. NELL has 134 classes and 1,184,377 instances whereas DBpedia has 138 classes and 631,461 instances.

ATMatcher is the second best matcher together with ALOD2Vec and Wiktionary with a F-Measure of 0.89. Only KGMatcher (0.94) could find more correct correspondences. For this track it would help to find classes based on the instances matches as already done by DOME matcher. The currently version of ATMatch only uses the classes to improve the instance correspondences. In the next version we plan to also add this component to increase the capabilities of this matcher.

2.7 Knowledge Graph

The results of ATBox are similar to previous years because the class hierarchy in this track is not deep. One possibility would be to use the categories (connected with property dcterms:subject) as an additional type of class information.

The F-Measure is 0.85 which is only slightly higher than the baseline using label and alternative label (0.84). Only ALOD2Vec and Wiktionary can improve on these results (both 0.87).

Regarding the runtime, ATMatcher is the fastest one with only 20 minutes for all test cases. Only the baselines are faster which need usually 11 minutes.

3 General comments

3.1 Discussions on the way to improve the proposed system

We would like to extend the matching pipeline with further components such as transformer[5] based comparison between a textual representation of resources. 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 [2].

4 Conclusions

In this paper, we have analyzed the results of ATBox matcher in OAEI 2021. It shows that the system is very scalable and can generate class, property and instance alignments.

Most of the used matching components are furthermore included in the MELT framework[4] to allow other system developers to reuse them.

5 http://purl.org/dc/terms/subject
References