






# MELT - Matching Evaluation Toolkit

Sven Hertling<sup>1</sup>, Jan Portisch<sup>1,2</sup>, and Heiko Paulheim<sup>1</sup>

<sup>1</sup> Data and Web Science Group, University of Mannheim, Mannheim, Germany  
{sven,jan,heiko}@informatik.uni-mannheim.de

<sup>2</sup> SAP SE Product Engineering Financial Services, Walldorf, Germany  
jan.portisch@sap.com

**Abstract.** In this paper, we present the Ontology Matching Evaluation Toolkit (MELT), a software toolkit to facilitate ontology matcher development, configuration, evaluation, and packaging. Compared to existing tools in the ontology matching domain, our framework offers detailed evaluation capabilities on the correspondence level of alignments as well as extensive group evaluation possibilities. A particular focus is put on a streamlined development and evaluation process along with ease of use for matcher developers and evaluators. Our contributions are twofold: We present an open source matching toolkit that integrates well into existing platforms, as well as an exemplary analysis of two OAEI 2018 tracks demonstrating advantages and analytical capabilities of MELT.

**Keywords:** Ontology matching · Evaluation framework · OAEI · SEALS · HOBBIT

## 1 Introduction

Ontology matching or ontology alignment is the non-trivial task of finding correspondences between entities of a set of given ontologies [10]. The matching can be performed manually or through the use of an automated matching system. For systematically evaluating the quality of such matchers, the Ontology Alignment Evaluation Initiative (OAEI) has been running campaigns [9] every year since 2005. Unlike other evaluation campaigns where researchers submit *data sets* as solutions to report their results (such as Kaggle<sup>1</sup>), the OAEI requires participants to submit a matching *system*, which is then executed on-site. After the evaluation, the results are publicly reported<sup>2</sup>. Therefore, execution and evaluation platforms have been developed and OAEI participants are required to package and submit their matching system for the corresponding platform. Two well-known platforms are used in the ontology matching community: The Semantic Evaluation at Large Scale (SEALS)<sup>3</sup> [12, 35] and the more recent Holistic Benchmarking of Big Linked Data (HOBBIT)<sup>4</sup> [24].

<sup>1</sup> <https://www.kaggle.com>.

<sup>2</sup> <http://oei.ontologymatching.org/2018/results/index.html>.

<sup>3</sup> <http://www.seals-project.eu>.

<sup>4</sup> <http://project-hobbit.eu>.

Based on the results of the OAEI 2018 campaign [1], only 4 out of 12 tracks were available in HOBBIT (*LargeBio*, *Link Discovery*, *SPIMBENCH*, *KnowledgeGraph*). Out of 19 matchers that were submitted in the 2018 campaign, only 6 matchers supported both, SEALS and HOBBIT, and 2 supported HOBBIT exclusively. The remaining 11 matchers supported only SEALS. While one reason for the low HOBBIT adoption might be its novelty, it also requires more steps to package a matcher for the HOBBIT platform and knowledge of the Docker<sup>5</sup> virtualization software. In particular for new entrants to the ontology matching community, the existing tooling might appear overwhelmingly complicated. In addition to potential obstacles for matcher development and submission, another observation from the OAEI campaigns is that the evaluation varies greatly among the different tracks that are offered e.g. *Anatomy* results contain *Recall+* as well as alignment coherence whereas the *Conference* track focuses on different reference alignments. Due to limited group evaluation capabilities in existing frameworks, some track organizers even developed their own evaluation systems.

For these reasons we present the *Matching Evaluation Toolkit* (MELT)<sup>6</sup> – an open source toolkit for ontology matcher development, fine-tuning, submission, and evaluation. The target audience are matching system developers as well as researchers who run evaluations on multiple matching systems such as OAEI track organizers. Likewise, system developers can use this tool to analyze the performance and errors of their systems in order to improve it. Furthermore, they can package and submit the system easily to OAEI campaigns.

The rest of this paper is structured as follows: Sect. 2 describes other work in the field of alignment visualization and evaluation. Section 3 gives an overview of the MELT framework and its possibilities whereas Sect. 4 shows an exemplary analysis of the latest systems submitted to the OAEI. We finish with an outlook on future developments.

## 2 Related Work

As MELT can be used both for evaluating ontology matching tools, as well as visualizing matching results, we discuss related works in both fields.

### 2.1 Matching and Alignment Evaluation Platforms

OAEI campaigns consist of multiple problem sets, so called *tracks*. Each track has its organizers who provide the datasets including reference alignments, execute the matching systems, and prepare the results page for the participants and the whole community. The track contains one or more test cases which correspond to a specific matching task consisting of two ontologies and a reference alignment. In 2010, three tracks (*Benchmark*, *Anatomy*, and *Conference*) were adjusted

<sup>5</sup> <https://www.docker.com>.

<sup>6</sup> <https://github.com/dwslab/melt>.

to be run with the SEALS platform [8]. One year later, participants of OAEI campaigns had to implement a matching interface and the SEALS client was the main tool used for executing and evaluating matchers. The interface contains a simple method (`align()`) which receives a URL for the source and a URL for the target ontology and has to return a URL which points to a file containing all correspondences in the alignment format<sup>7</sup>. This format is defined and used by the *Alignment API* [5].

Starting from 2017, a second evaluation platform, called *HOBBIT*, was added [18]. One difference compared to SEALS is that the system has to be submitted as a Docker image to a *GitLab* instance<sup>8</sup>, and in the corresponding project, a matcher description file has to be created. After submission of the matching system, the whole evaluation runs on servers of the *HOBBIT* platform. Thus, the source code for evaluating the matchers has to be submitted as a Docker image as well. All Docker containers communicate with each other over a message broker (*RabbitMQ*<sup>9</sup>). Hence, the interface between a system and the evaluation component can be arbitrary. To keep a similar interface to SEALS, the data generation component transfers two ontologies and the system adapter receives the URL to these files. It should return a file similar to the SEALS interface.

Working with alignments in Java code can be achieved with the *Alignment API* [5]. It is the most well-known API for ontology matching and can be used for loading and persisting alignments as well as for evaluating them with a set of possible evaluation strategies. Moreover, it provides some matching systems which are also used in OAEI campaigns as a baseline. Unfortunately, it is not yet enabled to be used with the maven build system<sup>10</sup>. Therefore, instead of using this API, some system developers created their own classes to work with alignments and to store them on disk<sup>11</sup> in order to be compatible with the evaluation interface.

*Alignment Visualization.* A lot of work has been done in the area of analyzing, editing, and visualizing alignments or ontologies with a graphical user interface. One example is *Alignment Cubes* [15], which allows an interactive visual exploration and evaluation of alignments. An advantage is the fine grained analysis on the level of an individual correspondence. It further allows to visualize the performance history of a matcher, for instance, which correspondences a matcher found in the most recent OAEI campaign but not in the previous one. Another framework for working with alignment files is *VOAR* [28, 29]. It is a Web-based system where users can upload ontologies and alignments. *VOAR* then allows the user to render them with multiple visualization types. The upload size of ontologies as well as alignments is restricted so that very large files cannot be uploaded.

<sup>7</sup> <http://alignapi.gforge.inria.fr/format.html>.

<sup>8</sup> <https://master.project-hobbit.eu>.

<sup>9</sup> <https://www.rabbitmq.com>.

<sup>10</sup> <https://maven.apache.org/>.

<sup>11</sup> [https://github.com/ernestojimenezruiz/logmap-matcher/tree/master/src/main/java/uk/ac/ox/krr/logmap\\_lite/io](https://github.com/ernestojimenezruiz/logmap-matcher/tree/master/src/main/java/uk/ac/ox/krr/logmap_lite/io).

Similar to *VOAR*, the *SILK workbench* [33] is also a Web-based tool with a focus on link/correspondence creation between different data sets in the *Linked Open Data Cloud*<sup>12</sup>. Unlike *VOAR*, it usually runs on the user’s computer. Matching operations (such as Levenshtein distance [20]) are visualized as nodes in a computation graph. The found correspondences are displayed and can be modified to further specify which concepts should be matched.

Further visualization approaches were pursued by matching system developers to actually fine-tune their systems. All these visualizations are therefore very specific to a particular matching approach. One such example is *YAM++* [23], which is a matching system based on a machine learning approach. Results are visualized in a split view where the class hierarchy of the two input ontologies is shown on each side lines are drawn between the matched classes. The user can modify the alignment with the help of this GUI. In a similar way, the developers of *COMA++* [2] created a user interface for their results. A visualization of whole ontologies is not implemented by the current tools but can be achieved with the help of *VOWL* [21] or *Web Protégé* [32], for instance.

Our proposed framework MELT allows for detailed and reusable analyses such as the ones presented in this section due to its flexible metrics and evaluators. An overview of the framework is presented in the following section.

### 3 Matching Evaluation Toolkit

MELT is a software framework implemented in Java which aims to facilitate matcher development, configuration, packaging, and evaluation. In this section, we will first introduce *Yet Another Alignment API*, an API for ontology alignment which is integrated into the framework. Afterwards, the matcher development process in MELT is introduced. Subsections 3.3 and 3.4 cover specific aspects of the framework that have not yet been explicitly addressed in the community: The implementation of matchers outside of the Java programming language Subsect. 3.3 and the chaining matching workflows Subsect. 3.4. After explaining the tuning component of the framework, this section closes with the matcher evaluation process in MELT.

#### 3.1 YAAA: Yet Another Alignment API

To allow for a simple development workflow, MELT contains *Yet Another Alignment API* (YAAA). It is similar to the *Alignment API* presented earlier but contains additional improvements such as maven support and arbitrary indexing possibilities of correspondence elements allowing queries such as “retrieve all correspondences with a specific source”. This is very helpful for a fast evaluation of large-scale test cases containing large reference or system alignments. The indexing is done with the *cqengine* library<sup>13</sup>. The API is, in addition, capable

<sup>12</sup> <https://lod-cloud.net>.

<sup>13</sup> <https://github.com/npgall/cqengine/>.

of serializing and parsing alignments. It also makes sure that all characters are escaped and that the resulting XML is actually parseable<sup>14</sup>. As explainability is still an open issue in the ontology matching community [7,34], YAAA also allows for extensions to correspondences and alignments. This means that additional information such as debugging information or human-readable explanations can be added. If there is additional information available in the alignment, it will also be printed by the default `CSVEvaluator` which allows for immediate consumption in the analysis and evaluation process and hopefully fosters the usage of additional explanations in the alignment format.

It is important to note that MELT does not require the usage of YAAA for parameter tuning, executing, or packaging a matcher – but also works with other APIs such as the *Alignment API*. This allows to evaluate matchers that were not developed using YAAA (see Sect. 4).

### 3.2 Matcher Development Workflow

In order to develop a matcher in Java with MELT, the first step is to decide which matching interface to implement. The most general interface is encapsulated in class `MatcherURL` which receives two URLs of the ontologies to be matched together with a URL referencing an input alignment. The return value should be a URL representing a file with correspondences in the alignment format. Since this interface is not very convenient, we also provide more specialized classes. In the `matching-yaaa` package we set the alignment library to YAAA. All matchers implementing interfaces from this package have to use the library and get at the same time an easier to handle interface of correspondences. In further specializations we also set the Semantic Web framework which is used to represent the ontologies. For a better usability, the two most well-known frameworks are integrated into MELT: *Apache Jena*<sup>15</sup> [3] (`MatcherYAAAJena`) and the *OWL API*<sup>16</sup> [14] (`MatcherYAAAOWlApi`). As the latter two classes are organized as separate maven projects, only the libraries which are actually required for the matcher are loaded. In addition, further services were implemented such as an ontology cache which ensures that ontologies are parsed only once. This is helpful, for instance, when the matcher accesses an ontology multiple times, when multiple matchers work together in a pipeline, or when multiple matchers shall be evaluated. We explicitly chose a framework-independent architecture so that developers can use the full functionality of the frameworks they already know rather than having to understand an additional wrapping layer. The different levels at which a matcher can be developed as well as how the classes presented in this section work together, are displayed in Fig. 1.

<sup>14</sup> This is not always the case for other implementations.

<sup>15</sup> <https://jena.apache.org>.

<sup>16</sup> <http://owlcs.github.io/owlapi/>.

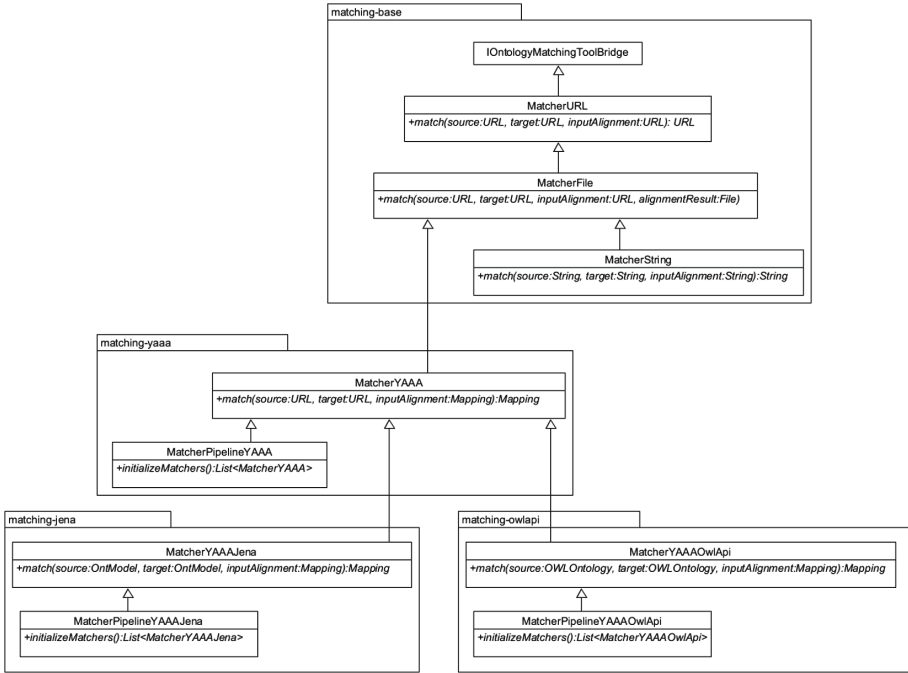


Fig. 1. Different possibilities to implement matchers

### 3.3 External Matching

The current ontology matching development and evaluation frameworks that are available focus on the Java programming language. As researchers apply advances in machine learning and natural language processing to other domains, they often turn to Python because leading machine learning libraries such as *scikit-learn*<sup>17</sup>, *TensorFlow*<sup>18</sup>, *PyTorch*<sup>19</sup>, *Keras*<sup>20</sup>, or *gensim* [26] are not easily available for the Java language. In the 2018 OAEI campaign, the first tools using such frameworks for ontology matching have been submitted [1].

To accommodate for the changes outlined, MELT allows to develop a matcher in any other programming language and wrap it as a SEALS or HOBBIT package. Therefore, class `MatcherExternal` has to be extended. It has to transform the given ontology URIs and input alignments to an executable command line call. The interface for the external process is simple. It receives the input variables via the command line and outputs the results via the standard output of the process – similar to many Unix command line tools. An example for a

<sup>17</sup> <https://scikit-learn.org/>.

<sup>18</sup> <https://www.tensorflow.org/>.

<sup>19</sup> <https://pytorch.org/>.

<sup>20</sup> <https://keras.io/>.

matcher implemented in Python is available on GitHub<sup>21</sup>. It also contains a simple implementation of the alignment format to allow Python matchers serializing their correspondences.

When executing the matcher with the SEALS client, the matching system is loaded into the Java virtual machine (JVM) of the SEALS client (evaluation code) with a customized class loader. This raises two points: (1) The code under test is executed in the same JVM and can probably access the code for evaluation. (2) The used class loader from the *JCL library*<sup>22</sup> does not implement all methods (specifically `getPackage()` and `getResource()`) of a class loader. However, these methods are used by other Java libraries<sup>23</sup> to load operating system dependent files contained in the jar file. Thus, some libraries do not work when evaluating a matcher with SEALS. Another problem is that all libraries used by the matching system may collide with libraries used by SEALS. This can cause issues with *Jena* and other Semantic Web frameworks because of the same JVM instance. To solve this issue, `MatcherExternal` can not only be used for matchers written in another programming language but also for Java matchers which use dependencies that are incompatible with the SEALS platform.

### 3.4 Pipelining Matchers

Ontology matchers often combine multiple matching approaches and sometimes consist of the same parts. An example would be a string-based matching of elements, and the application of a stable marriage algorithm or another matching refinement step on the resulting similarity matrix.

Following this observation, MELT allows for the chaining of matchers: The alignment of one matcher is then the input for the next matcher in the pipeline. The ontology caching services of MELT mentioned above prevent performance problems arising from repetitive loading and parsing of ontologies.

In order to execute a matcher pipeline, classes `MatcherPipelineYAAA` (for matchers that use different ontology management frameworks), `MatcherPipelineYAAAJena` (for pure *Jena* pipelines), and `MatcherPipelineYAAAOwlApi` (for pure *OWL API* pipelines) can be extended. Here the `initializeMatchers()` method has to be implemented. It returns matcher instances as a `List` in the order in which they shall be executed. These reusable parts of a matcher can easily be uploaded to GitHub to allow other developers to use common functionality<sup>24</sup>.

<sup>21</sup> <https://github.com/dwslab/melt/tree/master/examples/externalPythonMatcher>.

<sup>22</sup> <https://github.com/kamranzafar/JCL/blob/master/JCL/src/xeus/jcl/AbstractClassLoader.java>.

<sup>23</sup> An example would be class `SQLiteJDBCLoader` in `sqlite-jdbc` which uses these class loader methods.

<sup>24</sup> Other GitHub dependencies can be included by using <https://jitpack.io>, for instance.

### 3.5 Tuning Matchers

Many ontology matching systems require parameters to be set at design time. Those can significantly influence the matching system's performance. An example for a parameter would be the threshold parameter of a matcher utilizing a normalized string distance metric. For tuning such a system, MELT offers a `GridSearch` functionality. It requires a matcher and one or more parameters together with their corresponding search spaces, i.e. the values that shall be tested. The Cartesian product of these values is computed and each system configuration (an element of the Cartesian product which is a tuple of values) runs on the specified test case. The result is an `ExecutionResultSet` which can be further processed like any other result of matchers in MELT. To speed up the execution, class `Executor` was extended and can run matchers in parallel. Properties can be specified by a simple string. Therefore, the `JavaBeans` specification<sup>25</sup> is used to access the properties with so called setter-methods. This strategy allows also to change properties of nested classes or any list or map. An example of a matcher tuning can be found in the MELT repository<sup>26</sup>.

### 3.6 Evaluation Workflow

MELT defines a workflow for matcher execution and evaluation. Therefore, it utilizes the vocabulary used by the OAEI: A matcher can be evaluated on a `TestCase`, i.e. a single ontology matching task. One or more test cases are summarized in a `Track`. MELT contains a built-in `TrackRepository` which allows to access all OAEI tracks and test cases at design time without actually downloading them from the OAEI Web page. At runtime `TrackRepository` checks whether the required ontologies and alignments are available in the internal buffer; if data is missing, it is automatically downloading and caching it for the next access. The caching mechanism is an advantage over the SEALS platform which downloads all ontologies again at runtime which slows down the evaluation process if run multiple times in a row.

One or more matchers are given, together with the track or test case on which they shall be run, to an `Executor`. The `Executor` runs a matcher or a list of matchers on a single test case, a list of test cases, or a track. The `run()` method of the executor returns an `ExecutionResultSet`. The latter is a set of `ExecutionResult` instances which represent individual matching results on a particular test case. Lastly, an `Evaluator` accepts an `ExecutionResultSet` and performs an evaluation. Therefore, it may use one or more `Metric` objects. MELT contains various metrics, such as a `ConfusionMatrixMetric`, and evaluators. Nonetheless, the framework is designed to allow for the further implementation of evaluators and metrics.

<sup>25</sup> <https://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html>.

<sup>26</sup> [https://github.com/dwslab/melt/blob/master/examples/simpleJavaMatcher/src/test/java/de/uni\\_mannheim/informatik/dws/ontmatching/demomatcher/EvaluateMatcher.java](https://github.com/dwslab/melt/blob/master/examples/simpleJavaMatcher/src/test/java/de/uni_mannheim/informatik/dws/ontmatching/demomatcher/EvaluateMatcher.java).



After the `Executor` has run, an `ExecutionResult` can be refined by a `Refiner`. A refiner takes an individual `ExecutionResult` and makes it smaller. An example is the `TypeRefiner` which creates additional execution results depending on the type of the alignment (classes, properties, datatype properties, object properties, instances). Another example for an implemented refiner is the `ResidualRefiner` which only keeps non-trivial correspondences and can be used for metrics such as recall+. Refiners can be combined. This means that MELT can calculate very specific evaluation statistics such as the residual precision of datatype property correspondences.

A novelty of this framework is also the granularity at which alignments can be analyzed: The `EvaluatorCSV` writes every correspondence in a CSV format together with further details about the matched resources and the performed refinements. This allows for an in-depth analysis in various spreadsheet applications such as LibreOffice Calc where through the usage of filters analytical queries can be performed such as “false-positive datatype property matches by matcher X on test case Y”.

## 4 Exemplary Analysis of OAEI 2018 Results

In order to demonstrate the capabilities of MELT, a small analysis of the OAEI 2018 results for the *Conference* and *Anatomy* track has been performed and is presented in the following.

The *Conference* track consists of 16 ontologies from the conference domain. We evaluated all matching systems that participated in the 2018 campaign: *ALIN* [30], *ALOD2Vec* [25], *AML* [11], *DOME* [13], *FCAMapX* [4], *Holontology* [27], *KEPLER* [19], *Lily* [31], *LogMap* and *LogMapLt* [17], *SANOM* [22], as well as *XMap* [6].

The *Anatomy* track consists of a mapping between the human anatomy and the anatomy of a mouse. In the 2018 campaign, the same matchers mentioned above participated with the addition of *LogMapBio*, a matcher from the *LogMap* family [17].

First, the resulting alignments for *Anatomy*<sup>27</sup> and *Conference*<sup>28</sup> have been downloaded from the OAEI Web site. As both result sets follow the same structure every year, the MELT functions `Executor.loadFromAnatomyResultsFolder()` and `Executor.loadFromConferenceResultsFolder()` were used to load the results. The resulting `ExecutionResultSet` was then handed over to the `MatcherSimilarityMetric` and rendered using the `MatcherSimilarityLatexHeatMapWriter`. As the *Conference* track consists of multiple test cases, the results have to be averaged. Here, out of the available calculation modes in MELT, micro-average was chosen as this calculation mode is also used on the

<sup>27</sup> <http://oei.ontologymatching.org/2018/results/anatomy/oei2018-anatomy-alignments.zip>.

<sup>28</sup> <http://oei.ontologymatching.org/2018/conference/data/conference2018-results.zip>.

official results page<sup>29</sup> to calculate precision and recall scores. Altogether, the analysis was performed with few lines of Java code.<sup>30</sup>

Tables 1 and 2 show the Jaccard overlap [16] of the correspondences rendered as heat map where darker colors indicate a higher similarity. The Jaccard coefficient  $J \in [0, 1]$  between two alignments  $a_1$  and  $a_2$  with correspondences  $corr(a_1)$  and  $corr(a_2)$  was obtained as follows:

$$J(a_1, a_2) = \frac{|corr(a_1) \cap corr(a_2)|}{|corr(a_1) \cup corr(a_2)|}$$

In Table 1 it can be seen that – despite the various approaches that are pursued by the matching systems – most of them arrive at very similar alignments. One outlier in this statistic is *Holontology*. This is due to the very low number of correspondences overall found by this matching system (456 as opposed to ALIN, which had the second-smallest alignment with 928 matches).

Similarly, the matching systems of the *Conference* track also show commonalities in their alignments albeit the similarity here is less pronounced compared to the *Anatomy* track: The median similarity (excluding perfect similarities due to self-comparisons) of matching systems for *Anatomy* is  $median_{Anatomy} = 0.7223$  whereas the median similarity for *Conference* is  $median_{Conference} = 0.5917$ . The lower matcher similarity median indicates that *Conference* is a harder matching task because the matching systems have more disagreement about certain correspondences.

In a second step, the same result from the `MatcherSimilarityMetric` has been printed by another writer (`MatcherSimilarityLatexPlotWriter`) which plots the mean absolute deviation (MAD) on the X-axis and the  $F_1$  score on the Y-axis. MAD was obtained for each matcher by applying

$$MAD = \frac{1}{n} \sum_{i=1}^n |x_i - mean(X)|$$

where  $X$  is the set of Jaccard similarities for a particular matcher. The resulting plots are shown in Figs. 2 and 3. It can be seen that the matchers form different clusters: *Anatomy* matchers with a high  $F_1$  measure have also a high deviation. Consequently, those matchers are likely candidates for a combination to achieve better results. On *Conference*, on the other hand, good combinations cannot be derived because the best matchers measured by their  $F_1$  score tend not to deviate much in their resulting alignments.

In addition to the evaluations performed using the matcher similarity metric, the `EvaluatorCSV` was run using the OAEI 2018 matchers on the *Anatomy* and *Conference* tracks. The resulting CSV file contains one row for each correspondence together with additional information about each resource that is mapped

<sup>29</sup> <http://oei.ontologymatching.org/2018/results/conference/>.

<sup>30</sup> The code to run the analysis can be found on GitHub: <https://github.com/dwslab/melt/tree/master/examples/analyzingMatcherSimilarity>.

**Table 1.** OAEI anatomy 2018 alignment similarity

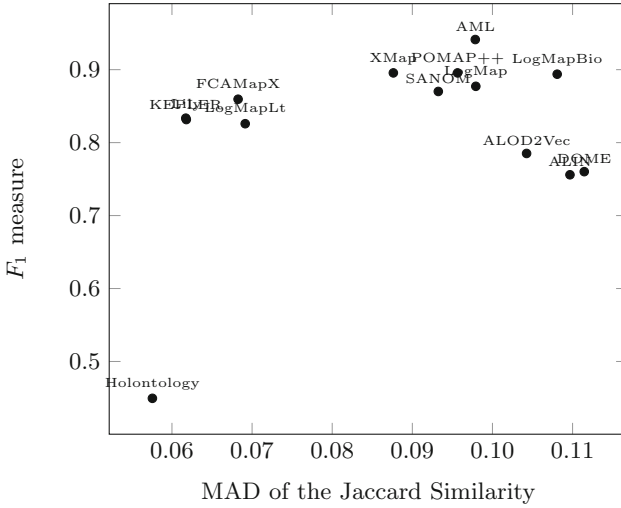
	ALIN	ALOD2Vec	AML	DOME	FCAMapX	Holontology	KEPLER	Lily	LogMap	LogMapBio	LogMapLt	POMAP++	SANOM	XMap
ALIN	1	0.93	0.62	0.97	0.72	0.47	0.79	0.63	0.66	0.6	0.81	0.63	0.62	0.65
ALOD2Vec	0.93	1	0.65	0.94	0.77	0.45	0.81	0.67	0.7	0.63	0.84	0.66	0.64	0.68
AML	0.62	0.65	1	0.62	0.76	0.3	0.74	0.72	0.8	0.82	0.72	0.83	0.79	0.83
DOME	0.97	0.94	0.62	1	0.73	0.47	0.79	0.64	0.66	0.6	0.81	0.63	0.62	0.66
FCAMapX	0.72	0.77	0.76	0.73	1	0.35	0.75	0.69	0.82	0.77	0.89	0.77	0.75	0.78
Holontology	0.47	0.45	0.3	0.47	0.35	1	0.38	0.3	0.32	0.29	0.39	0.31	0.3	0.31
KEPLER	0.79	0.81	0.74	0.79	0.75	0.38	1	0.69	0.78	0.72	0.75	0.76	0.71	0.76
Lily	0.63	0.67	0.72	0.64	0.69	0.3	0.69	1	0.7	0.68	0.69	0.72	0.72	0.72
LogMap	0.66	0.7	0.8	0.66	0.82	0.32	0.78	0.7	1	0.9	0.81	0.81	0.8	0.81
LogMapBio	0.6	0.63	0.82	0.6	0.77	0.29	0.72	0.68	0.9	1	0.74	0.8	0.78	0.78
LogMapLt	0.81	0.84	0.72	0.81	0.89	0.39	0.75	0.69	0.81	0.74	1	0.74	0.74	0.75
POMAP++	0.63	0.66	0.83	0.63	0.77	0.31	0.76	0.72	0.81	0.8	0.74	1	0.79	0.83
SANOM	0.62	0.64	0.79	0.62	0.75	0.3	0.71	0.72	0.8	0.78	0.74	0.79	1	0.78
XMap	0.65	0.68	0.83	0.66	0.78	0.31	0.76	0.72	0.81	0.78	0.75	0.83	0.78	1

**Table 2.** OAEI conference 2018 alignment similarity

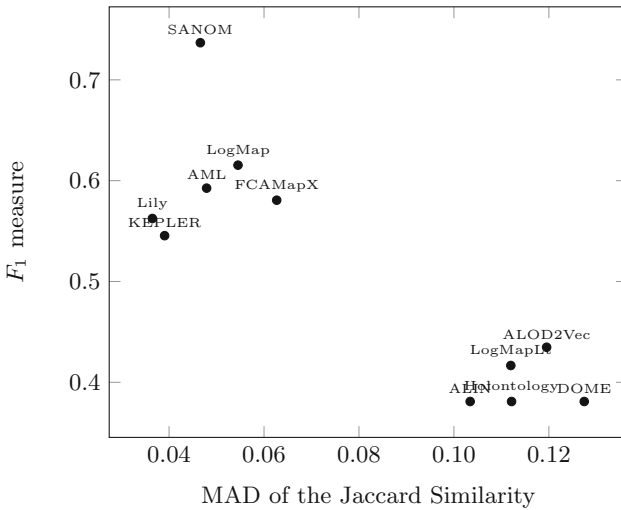
	ALIN	ALOD2Vec	AML	DOME	FCAMapX	Holontology	KEPLER	Lily	LogMap	LogMapLt	SANOM	XMap
ALIN	1	0.75	0.65	0.84	0.63	0.77	0.53	0.43	0.72	0.76	0.52	0.6
ALOD2Vec	0.75	1	0.58	0.87	0.67	0.75	0.61	0.37	0.67	0.86	0.5	0.54
AML	0.65	0.58	1	0.61	0.58	0.56	0.53	0.45	0.71	0.59	0.63	0.64
DOME	0.84	0.87	0.61	1	0.67	0.81	0.59	0.39	0.7	0.86	0.52	0.56
FCAMapX	0.63	0.67	0.58	0.67	1	0.6	0.55	0.41	0.62	0.66	0.51	0.53
Holontology	0.77	0.75	0.56	0.81	0.6	1	0.53	0.37	0.64	0.72	0.49	0.52
KEPLER	0.53	0.61	0.53	0.59	0.55	0.53	1	0.41	0.57	0.62	0.5	0.54
Lily	0.43	0.37	0.45	0.39	0.41	0.37	0.41	1	0.46	0.39	0.48	0.51
LogMap	0.72	0.67	0.71	0.7	0.62	0.64	0.57	0.46	1	0.7	0.63	0.66
LogMapLt	0.76	0.86	0.59	0.86	0.66	0.72	0.62	0.39	0.7	1	0.51	0.56
SANOM	0.52	0.5	0.63	0.52	0.51	0.49	0.5	0.48	0.63	0.51	1	0.61
XMap	0.6	0.54	0.64	0.56	0.53	0.52	0.54	0.51	0.66	0.56	0.61	1

(e.g. label, comment, or type) and with additional information about the correspondence itself (e.g. residual match indicator or evaluation result). All files are available online for further analysis on correspondence level.<sup>31</sup>

<sup>31</sup> <https://github.com/dwslab/melt/tree/master/examples/analyzingMatcherSimilarity>.



**Fig. 2.** Matcher comparison using MAD and  $F_1$  on the *Anatomy* data set



**Fig. 3.** Matcher comparison using MAD and  $F_1$  on the *Conference* data set

## 5 Conclusion

With MELT, we have presented a framework for ontology matcher development, configuration, packaging, and evaluation. We hope to lower the entrance barriers into the ontology matching community by offering a streamlined development process. MELT can also simplify the work of researchers who evaluate multiple

matchers on multiple data sets such as OAEI track organizers through its rich evaluation capabilities.

The evaluation capabilities were exemplarily demonstrated for two OAEI tracks by providing a novel view on matcher similarity. The MELT framework as well as the code used for the analyses presented in this paper are open-source and freely available.

Future work will focus on adding more evaluation possibilities in the form of further refiners and reasoners, providing more default matching functionalities such as modular matchers that can be used in matching pipelines, and developing visual evaluation support based on the framework to allow for better ontology matcher results comparisons.

## References

1. Algergawy, A., et al.: Results of the ontology alignment evaluation initiative 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 76–116 (2018). CEUR-WS.org
2. Aumüller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 906–908. ACM (2005)
3. Carroll, J., Reynolds, D., Dickinson, I., Seaborne, A., Dollin, C., Wilkinson, K.: Jena: implementing the semantic web recommendations. In: Proceedings of the 13th International World Wide Web (WWW) Conference, pp. 74–83. ACM, New York (2004)
4. Chen, G., Zhang, S.: FCAMapX results for OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 160–166 (2018). CEUR-WS.org
5. David, J., Euzenat, J., Scharffe, F., Trojahn dos Santos, C.: The alignment API 4.0. *Semant. Web J.* **2**(1), 3–10 (2011)
6. Djeddi, W.E., Yahia, S.B., Khadir, M.T.: XMap: results for OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 210–215 (2018). CEUR-WS.org
7. Dragisic, Z., Ivanova, V., Lambrix, P., Faria, D., Jiménez-Ruiz, E., Pesquita, C.: User validation in ontology alignment. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9981, pp. 200–217. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-46523-4\\_13](https://doi.org/10.1007/978-3-319-46523-4_13)
8. Euzenat, J., et al.: Results of the ontology alignment evaluation initiative 2011. In: OM, CEUR Workshop Proceedings, vol. 814 (2011). CEUR-WS.org
9. Euzenat, J., Meilicke, C., Stuckenschmidt, H., Shvaiko, P., Trojahn, C.: Ontology alignment evaluation initiative: six years of experience. In: Spaccapietra, S. (ed.) *Journal on Data Semantics XV*. LNCS, vol. 6720, pp. 158–192. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22630-4\\_6](https://doi.org/10.1007/978-3-642-22630-4_6)
10. Euzenat, J., Shvaiko, P.: *Ontology Matching*, 2nd edn. Springer, New York (2013)
11. Faria, D., et al.: Results of AML participation in OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 125–131 (2018). CEUR-WS.org
12. García-Castro, R., Esteban-Gutiérrez, M., Gómez-Pérez, A.: Towards an infrastructure for the evaluation of semantic technologies. In: *eChallenges e-2010 Conference*, pp. 1–7. IEEE (2010)
13. Hertling, S., Paulheim, H.: DOME results for OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 144–151 (2018). CEUR-WS.org

14. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 touch paper: the OWL API. In: OWLED 258 (2007)
15. Ivanova, V., Bach, B., Pietriga, E., Lambrix, P.: Alignment cubes: towards interactive visual exploration and evaluation of multiple ontology alignments. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 400–417. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-68288-4\\_24](https://doi.org/10.1007/978-3-319-68288-4_24)
16. Jaccard, P.: Lois de distribution florale dans la zone alpine. *Bull. Soc. Vaudoise Sci. Nat.* **38**, 69–130 (1902). <https://doi.org/10.5169/seals-266762>
17. Jiménez-Ruiz, E., Grau, B.C., Cross, V.: Logmap family participation in the OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 187–191 (2018). CEUR-WS.org
18. Jiménez-Ruiz, E., et al.: Introducing the HOBBIT platform into the ontology alignment evaluation campaign. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 49–60 (2018). CEUR-WS.org
19. Kachroudi, M., Diallo, G., Yahia, S.B.: KEPLER at OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 173–178 (2018). CEUR-WS.org
20. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* **10**(8), 707–710 (1966)
21. Lohmann, S., Negru, S., Haag, F., Ertl, T.: Visualizing ontologies with VOWL. *Semant. Web* **7**(4), 399–419 (2016). <https://doi.org/10.3233/SW-150200>
22. Mohammadi, M., Hofman, W., Tan, Y.: SANOM results for OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 205–209 (2018). CEUR-WS.org
23. Ngo, D.H., Bellahsene, Z.: YAM++: a multi-strategy based approach for ontology matching task. In: ten Teije, A., et al. (eds.) EKAW 2012. LNCS (LNAI), vol. 7603, pp. 421–425. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-33876-2\\_38](https://doi.org/10.1007/978-3-642-33876-2_38)
24. Ngomo, A.C.N., Röder, M.: HOBBIT: holistic benchmarking for big linked data. In: ERCIM News, no. 105 (2016)
25. Portisch, J., Paulheim, H.: ALOD2Vec matcher. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 132–137 (2018). CEUR-WS.org
26. Řehůřek, R., Sojka, P.: Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, ELRA, Valletta, Malta, pp. 45–50, May 2010. <http://is.muni.cz/publication/884893/en>
27. Roussille, P., Megdiche, I., Teste, O., Trojahn, C.: Holontology: results of the 2018 OAEI evaluation campaign. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 167–172 (2018). CEUR-WS.org
28. Severo, B., dos Santos, C.T., Vieira, R.: VOAR: a visual and integrated ontology alignment environment. In: Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, 26–31 May 2014, pp. 3671–3677 (2014)
29. Severo, B., Trojahn, C., Vieira, R.: VOAR 3.0 : a configurable environment for manipulating multiple ontology alignments. In: International Semantic Web Conference (Posters, Demos & Industry Tracks), CEUR Workshop Proceedings, vol. 1963 (2017)
30. da Silva, J., Revoredo, K., Baião, F.A.: ALIN results for OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 117–124 (2018). CEUR-WS.org
31. Tang, Y., Wang, P., Pan, Z., Liu, H.: Lily results for OAEI 2018. In: OM@ISWC, CEUR Workshop Proceedings, vol. 2288, pp. 179–186 (2018). CEUR-WS.org

32. Tudorache, T., Vendetti, J., Noy, N.F.: Web-protege: a lightweight OWL ontology editor for the web. In: OWLED, CEUR Workshop Proceedings, vol. 432 (2008). CEUR-WS.org
33. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Silk - a link discovery framework for the web of data. In: LDOW, vol. 538 (2009)
34. Wang, X., Haas, L., Meliou, A.: Explaining data integration. *Data Eng. Bull.* **41**(2), 47–58 (2018)
35. Wrigley, S.N., García-Castro, R., Nixon, L.: Semantic evaluation at large scale (SEALS). In: Proceedings of the 21st International Conference Companion on World Wide Web - WWW 2012 Companion, pp. 299–302. ACM Press, Lyon (2012)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

