YAM++ – Results for OAEI 2012*

DuyHoa Ngo, Zohra Bellahsene

University Montpellier 2, INRIA, LIRMM
{firstname.lastname@lirmm.fr}

Abstract. The YAM++ system is a self configuration, flexible and extensible ontology matching system. YAM++ takes advantages of many techniques coming from different fields such as machine learning, information retrieval, graph matching, etc. in order to enhance the matching quality. In this paper, we briefly present the YAM++ approach and its results on OAEI 2012 campaign.

1 Presentation of the system

YAM++ - (not) Yet Another Matcher is an automatic, flexible and self-configuring ontology matching system for discovering semantic correspondences between entities (i.e., classes, object properties and data properties) of ontologies. In YAM++ approach, multiple working strategies and matching techniques coming from machine learning, information retrieval, graph matching have been implemented in order to deal with both terminological and conceptual heterogeneity of ontologies. In the past, YAM++ achieved good results and gained high ranking positions in comparison with other participants in Benchmark, Conference and Multifarm tracks in OAEI 2011 and OAEI 2011.5 campaigns. This year, YAM++ participates in **six tracks** including **Benchmark**, **Conference**, **Multifarm**, **Library**, **Anatomy** and **Large Biomedical Ontologies** tracks.

1.1 State, purpose, general statement

The major principle of the matching strategy in YAM++ approach is utilizing as much useful information as possible of entities in ontologies effectively and efficiently. In the previous YAM++ (OAEI **2011** version), it is a combination of machine learning and graph matching techniques. In particular, Decision Tree learning model is used to combine different terminological similarity measures, whereas, a similarity propagation method is used to discover mappings by exploiting structural information of entities. The drawback of the previous version of YAM++ lies in its low performance in terms of time and high memory consuming. Therefore, it was inapplicable for large scale ontology matching scenarios.

In the current version (OAEI **2012**), several changes of YAM++ have been done. Firstly, since OAEI 2011.5 campaign, we have proposed new similarity measures based on techniques coming from information retrieval field in order to compare short and long texts. These measures are an alternative solution to the machine learning method,

* Supported by ANR DataRing ANR-08-VERSO-007-04.

which was used in the YAM++ 2011 version, in the case where no training data is available. Next, a semantic verification component have been added in YAM++ in order to enhance the matching quality. Finally, a candidate filtering component have been designed for reducing computational space when dealing with large scale ontology matching scenarios.

1.2 Specific techniques used

In this section, we will briefly describe the workflow of YAM++ and its main components, which are shown in Fig.1.

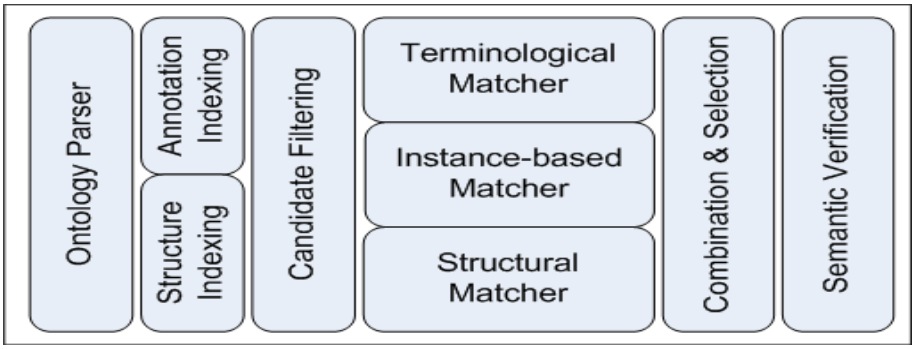

Fig. 1. Main components of YAM++ system

In YAM++ approach, a generic workflow for a given ontology matching scenario is as follows.

1. Input ontologies are loaded and parsed by a **Ontology Parser** component;
2. Information of entities in ontologies are indexed by the **Annotation Indexing** and the **Structure Indexing** components;
3. **Candidates Filtering** component filters out all possible pairs of entities from the input ontologies, whose descriptions are highly similar;
4. Among those candidate mappings, the **Terminological Matcher** component produces a set of mappings by comparing the annotations of entities;
5. The **Instance-based Matcher** component supplements new mappings through shared instances between ontologies;
6. In YAM++, matching results of the **Terminological Matcher** and the **Instance-based Matcher** are aggregated into an element level matching result. The **Structural Matcher** component then enhances element level matching result by exploiting structural information of entities;
7. The mapping results obtained from the three matchers above are then combined and selected by the **Combination & Selection** component to have a unique set of mappings;
8. Finally, the **Semantic Verification** component refines those mappings in order to eliminate the inconsistent ones.

Ontology Parser To read and parse input ontologies, YAM++ uses OWLAPI open source library. In addition, YAM++ makes use of Pellet - an OWL 2 Reasoner in order to discover hidden relations between entities in ontologies. Here, the whole ontology is stored in the main memory.

Annotation Indexing In this component, all annotations information of entities such as ID, labels and comments are extracted. The languages used for representing annotations are considered. In the case where input ontologies use different languages to describe the annotations of entities, a multilingual translator (**Microsoft Bing**) is used to translate those annotations to English. Those annotations are then normalized by tokenizing into set of tokens, removing stop words, and stemming. Next, tokens are indexed in a table for future use.

Structure Indexing In this component, the main structure information such as IS-A and PART-OF hierarchies of ontologies are stored. In particular, YAM++ assigns a compressed bitset values for every entity of the ontologies. Through the bitset values of each entity, YAM++ can fast and easily gets its ancestors, descendants, etc. A benefit of this method is to easily access to the structure information of ontolgy and minimize memory for storing it. After this step, the loaded ontologies can be released to save main memory.

Candidates Filtering The aim of this component is to reduce the computational space for a given scenario, especially for the large scale ontology matching tasks. In YAM++, two filters have been designed for the purpose of performing terminology-based matchers efficiently.

- A **Description Filter** is a search-based filter, which filters out candidate mappings before computing the real similarity values between the description of entities. Here, a description of an entity consists of its labels, synonym labels and comments. The idea of this filter is as follows. Firstly, the descriptions of all entities in the bigger size ontology are indexed by **Lucene** search engine. For each entity in the smaller size ontology, a multiple terms query created by tokens within the description of this entity is executed in order to find the **top-K** similar entities.
- A **Label Filter** is used to fast detect candidate mappings, where labels of entities in each candidate mapping are similar or differ in maximum two tokens. The intuition is that if two labels of two entities differ by more than three tokens, any string-based method will produce a low similarity score value. Then, these entities are highly unmatched.

Terminological Matcher In YAM++, the **Terminological Matcher** component is compounded by two sub-matcher namely **Label Matcher** and **Context Profile Matcher**.

- The **Label Matcher** splits all labels of entities into tokens and calculates the information content of each token in the whole ontology. Then, it makes use of Tversky similarity measure to compute similarity score between labels of entities. Let we

explain how this method works by an example with two entities: `cmt.owl#`*Co-author* and `conference.owl#`*Contribution_co-author*. After splitting and normalizing labels, we have 2 sets of tokens such as: {`coauthor`} and {`coauthor`, `contribution`}. Token `coauthor` appears in each input ontology only one time, whereas, token `contribution` appears 10 times among 60 concepts in the ontology `conference.owl`. Therefore, the information content of the token `contribution` is less than that of the token `coauthor`. In particular, the normalized TFIDF weights of each token inside the input ontologies are equal: $\{w_{coauthor} = 1.0\}$, $\{w_{coauthor} = 1.0, w_{contribution} = 0.34\}$. Two sets of tokens share only token `coauthor`, then the similarity computed by Tversky method is $\frac{1.0+1.0}{1.0+1.0+0.34} = 0.855$.

- The **Context Profile Matcher** is used to compute similarity value of entities by comparing their context profiles, which are normally a long text. Like in the first YAM++ version, a context profile of an entity may be an **Individual Profile**, **Semantic Profile** or **External Profile**. We refer to [4] for more detail about the construction of these profiles and computation of the similarity between them.

Instance-based Matcher In many situation, the input ontologies provide data (instances), therefore, the aim of the **Instance-based Matcher** is to discover new mappings which are complement to the result obtained from the **Terminological Matcher**. Basically, the **Instance-based Matcher** is not changed in from the first YAM++ version to the current version. Therefore, for saving space, we refer to section **Extensional Matcher** of [3] for more detail.

Structural Matcher The **Structural Matcher** component contains two similarity propagation methods namely Similarity Propagation and Confidence Propagation.

- **Similarity Propagation** method is a graph matching method, which inherits the main features of the well-known **Similarity Flooding** algorithm [2]. The only difference is about transforming an ontology to a directed labeled graph. This matcher is not changed from the first YAM++ version to the current version. Therefore, for saving space, we refer to section **Similarity Flooding** of [3] for more detail.
- The intuition of the **Confidence Propagation** method is as follows. Assume $\langle a_1, b_1, \equiv, c_1\rangle$ and $\langle a_2, b_2, \equiv, c_2\rangle$ are two initial mappings, which are maybe discovered by the element level matcher (i.e., terminological matcher or instance-based matcher). If a_1 and b_1 are ancestors of a_2 and b_2 respectively, then after running confidence propagation, we have $\langle a_1, b_1, \equiv, c_1 + c_2\rangle$ and $\langle a_2, b_2, \equiv, c_2 + c_1\rangle$. Note that, confidence values are propagated only among collection of initial mappings.

In YAM++, the aim of the **Similarity Propagation** method is discovering new mappings by exploiting as much as possible the structural information of entities. This method is used for a small scale ontology matching task, where the total number of entities in each ontology is smaller than 1000. In contrary, the **Confidence Propagation** method supports a **Semantic Verification** component to eliminate inconsistent mappings. This method is mainly used in a large scale ontology matching scenario.

Mappings Combination and Selection The aim of the **Mappings Combination and Selection** component is to produce a unique set of mappings from matching results obtained by terminological matcher, instance-based matcher and structural matcher. In this component, a **Dynamic Weighted Aggregation** method have been implemented. Given an ontology matching scenario, it automatically computes a weight value for each matcher and establishes a threshold value for selecting the best candidate mappings. The main idea of this method can be seen in [3] for more detail.

Semantic Verification After running the similarity or confidence propagation on overall candidate mappings, the final achieved similarity values reach a certain stability. Based on those values, YAM++ is able to remove inconsistent mappings with more certainty. There are two main steps in the **Semantic Verification** component such as (i) identifying inconsistent mappings, and (ii) elimination inconsistent mappings.

In order to identify inconsistencies, several semantic conflict patterns have been designed in YAM++ as follows:

- Two mappings $\langle a_1, b_1 \rangle$ and $\langle a_2, b_2 \rangle$ are crisscross conflict if a_1 is an ancestor of a_2 in ontology O_1 and b_2 is an ancestor of b_1 in ontology O_2.
- Two mappings $\langle a_1, b_1 \rangle$ and $\langle a_2, b_2 \rangle$ are disjointness subsumption conflict if a_1 is an ancestor of a_2 in ontology O_1 and b_2 disjoints with b_1 in ontology O_2.
- A property-property mapping $\langle p_1, p_2 \rangle$ is inconsistent with respect to alignment A if $\{Doms(p_1) \times Doms(p_2)\} \cap A = \emptyset$ and $\{Rans(p_1) \times Rans(p_2)\} \cap A = \emptyset$ then (p_1, p_2), where $Doms(p)$ and $Rans(p)$ return a set of domains and ranges of property p.
- Two mappings $\langle a, b_1 \rangle$ and $\langle a, b_2 \rangle$ are duplicated conflict if the cardinality matching is 1:1 (for a small scale ontology matching scenario) or the semantic similarity $SemSim(b_1, b_2)$ is less than a threshold value θ (for a large scale matching with cardinality 1:m).

In order to eliminate inconsistent mappings, a **Greedy Selection** method is used. The idea of this method is that it iteratively selects the mapping with the highest confidence value, which does not conflict with the mappings already selected before.

In YAM++, we used **Alcomo** [1] - an effective open source tool to eliminate inconsistent mappings for the first three conflict patterns. For the last pattern, a supplementary method called **Duplicate Removing** have been implemented. In this method, semantic similarity of two classes in ontology is computed by Resnik method [5], where an information content value of a class is computed by an intrinsic method described in [6].

1.3 Adaptations made for the evaluation

Before running the matching process, YAM++ analyzes the input ontologies and adapts itself to the matching task. In particular, if annotations of entities in input ontologies are described by different languages, YAM++ automatically translates them in English. If the number of entities in input ontologies is smaller than 1000, YAM++ is switched to small scale matching regime, otherwise, it runs with large scale matching regime. The main difference between two regime lies in the **Structural Matcher** and **Semantic Verification** components as we discussed above.

1.4 Link to the system and parameters file

A SEALS client wrapper for YAM++ system and parameter files can be download at: http://www2.lirmm.fr/d̃ngo/YAMplusplus2012.zip. See the instructions in tutorial from SEALS platform[1] to test our system.

1.5 Link to the set of provided alignments (in align format)

The results of all tracks can be downloaded at: http://www2.lirmm.fr/d̃ngo/ YAMplusplus2012Results.zip.

2 Results

In this section, we present the evaluation results obtained by running YAM++ with SEALS client with **Benchmark**, **Conference**, **Multifarm**, **Library**, **Anatomy** and **Large Biomedical Ontologies** tracks. All experiments are executed by YAM++ with SEALS client version 4.1 beta and JDK 1.6 on PC Intel 3.0 Pentium, 3Gb RAM, Window XP SP3.

2.1 Benchmark

In OAEI 2012, because Benchmark is a blind test, we do not have the complete results for all test sets. Table 1 shows the result of YAM++ running on the **pre-test** data set.

Test set	H-mean Precision	H-mean Recall	H-mean Fmeasure
Biblio	0.972	0.794	0.874
Jerm	0.988	0.967	0.978
Provenance	0.979	0.641	0.774
Finance	0.979	0.798	0.879

Table 1. YAM++ results on pre-test Benchmark track

2.2 Conference

Conference track now contains 16 ontologies from the same domain (conference organization) and each ontology must be matched against every other ontology. This track is an open+blind, so in the Table 2, we can only report our results with respect to the available reference alignments

Test set	H-mean Precision	H-mean Recall	H-mean Fmeasure
Conference	0.802	0.692	0.743

Table 2. YAM++ results on Conference track

[1] http://oaei.ontologymatching.org/2012/seals-eval.html

2.3 MultiFarm

The goal of the MultiFarm track is to evaluate the ability of matcher systems to deal with multilingual ontologies. It is based on the OntoFarm dataset, where annotations of entities are represented in different languages such as: English (en), Chinese (cn), Czech (cz), Dutch (nl), French (fr), German (de), Portuguese (pt), Russian (ru) and Spanish (es). For saving space, we do not list all results here. Instead, the results of YAM++ can be found at at SEALS result repository[2].

2.4 Anatomy

The Anatomy track consists of finding an alignment between the Adult Mouse Anatomy (2744 classes) and a part of the NCI Thesaurus (3304 classes) describing the human anatomy. Table 3 shows the evaluation result and runtime of YAM++ on this track.

Test set	Precision	Recall	Fmeasure	Run times
Anatomy	0.944	0.868	0.904	201 (s)

Table 3. YAM++ results on Anatomy track

2.5 Library

The library track is a real-word task to match the STW (6575 classes) and the TheSoz (8376 classes) thesaurus. Table 4 shows the evaluation result and runtime of YAM++ against an existing reference alignment on this track.

Test set	Precision	Recall	Fmeasure	Run times
Library	0.595	0.750	0.663	759 (s)

Table 4. YAM++ results on Library track

2.6 Large Biomedical Ontologies

This track consists of finding alignments between the Foundational Model of Anatomy (FMA), SNOMED CT, and the National Cancer Institute Thesaurus (NCI). There are 9 sub tasks with different size of input ontologies, i.e., small fragment, large fragment and the whole ontologies. Table 5 shows the evaluation results and run times of YAM++ on those sub tasks.

3 General comments

This is the third time YAM++ participates to the OAEI campaign. We found that SEALS platform is a very valuable tool to compare the performance of our system with the others. Besides, we also found that OAEI tracks covers a wide range of heterogeneity in ontology matching task. They are very useful to help developers/researchers to develop their semantic matching system.

[2] http://www.seals-project.eu/

Test set	Precision	Recall	Fmeasure	Run times
Small FMA - NCI	0.980	0.848	0.9093	482 (s)
Large FMA - NCI	0.923	0.821	0.869	1908 (s)
Whole FMA - NCI	0.906	0.821	0.861	3864 (s)
Small FMA - SNOMED	0.972	0.693	0.809	1990 (s)
Large FMA - SNOMED	0.879	0.684	0.769	7709 (s)
Whole FMA - SNOMED	0.878	0.683	0.768	9907 (s)
Small SNOMED - NCI	0.951	0.604	0.739	5643 (s)
Large SNOMED - NCI	0.864	0.599	0.708	13233 (s)
Whole SNOMED - NCI	0.859	0.599	0.706	17690 (s)

Table 5. YAM++ results on Large Biomedical Ontologies track

3.1 Comments on the results

The current version of YAM++ has shown a significant improvement in terms of matching quality and runtime with respect to the previous version. In particular, the H-mean Fmeasure value of the Conference track increases $0.74 - 0.65 = 0.09$; this version is able to run with not only scalability dataset but also very large scale dataset (i.e., Library, Biomedical ontologies).

4 Conclusion

In this paper, we have presented our ontology matching system called YAM++ and its evaluation results on different tracks on OAEI 2012 campaign. The experimental results are promising and show that YAM++ is able to work effectively and efficiently with real-world ontology matching tasks. In near future, we continue improving the matching quality and efficiency of YAM++. Furthermore, we plan to deal with instance matching track also.

References

1. Christian Meilicke. Alignment incoherence in ontology matching phd. thesis. In *University of Mannheim, Chair of Artificial Intelligence*, 2011.
2. Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.
3. DuyHoa Ngo, Zohra Bellahsene, and Remi Coletta. Yam++ results for oaei 2011. In *OM*, 2011.
4. DuyHoa Ngo, Zohra Bellasene, and Remi Coletta. A generic approach for combining linguistic and context profile metrics in ontology matching. In *ODBASE Conference*, 2011.
5. Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, 1995.
6. David Sánchez, Montserrat Batet, Aida Valls, and Karina Gibert. Ontology-driven web-based semantic similarity. *J. Intell. Inf. Syst.*, pages 383–413, 2010.