ODGOMS - Results for OAEI 2013*

I-Hong Kuo[1,2], Tai-Ting Wu[1,3]

[1] Industrial Technology Research Institute, Taiwan
[2] yihonguo@itri.org.tw
[3] taitingwu@itri.org.tw

Abstract. ODGOMS is a multi-strategy ontology matching system which consists of elemental level, structural level, and optimization level strategies. When it starts to match ontologies, it first exploits appropriate string-based and token-based similarity computing strategies to find preliminary aligned results, and then it filters these results and merges them by using the optimization strategies. Despite ODGOMS uses simple matching logic, the results show that it is competitive with other well known ontology matching tools.

1 Presentation of the system

1.1 State, purpose, general statement

ODGOMS (Open Data Group Ontology Matching System) is an ontology matching system exploited by our looking forward research plan in the company. The target of mentioned above research plan is to offer people an user-friendly integrated interface to search and to browse linked open data on the internet.

The main idea of ODGOMS is to exploit simple but useful matching and merging strategies to produce robust aligned results. All strategies used in the system can be grouped into three groups that are elemental level strategies, structural level strategies, and optimization level strategies.

We have submitted two versions of ODGOMS which are version 1.1 and version 1.2 to participate in OAEI 2013 campaign. Of the two the latter is better than the former. This is because the latter has fixed some bugs existed in the former and has added some new features. Since ODGOMS version 1.2 is the latest version of the system, we only describe the contents of it in the following sections.

1.2 Specific techniques used

ODGOMS focuses on developing individual ontology matching modules for different matching aspects and on finding an appropriate way to merge all matching modules.

* Supported by the looking-forward research plan in Industrial Technology Research Institute. The mentioned above plan is named "Data Refining for LOD Using Linked Data Integration Technology."

Each matching module of ODGOMS can be exploited individually by setting filter threshold and the positions of input ontologies. The system architecture of ODGOMS is shown in Fig. 1.

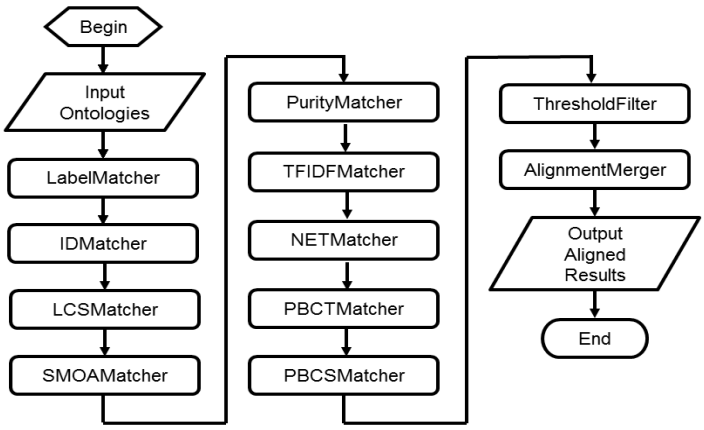

Fig. 1. System architecture of ODGOMS

The workflow of ODGOMS shown in Fig. 1 is described as follows. It first reads input ontologies into the memory, then it runs all matching modules individually which are LabelMatcher, IDMatcher, LCSMatcher, SMOAMatcher, PurityMatcher, TFIDFMatcher, NETMatcher, PBCTMatcher, and PBCSMatcher. After that it uses a filtering module named ThresholdFilter to filter all aligned results stored in each matching module, and merges them in an special order by exploiting an optimizating module named AlignmentMerger. At last, it outputs the integrated aligned results. All modules are divided into three groups which are elemental level modules, structural level modules, and optimization level modules. The detailed description of mentioned above modules are described as follows.

1.2.1 Elemental Level Modules

LabelMatcher For each entity in the first input ontology, this module finds a best matched entity in the second input ontology that has at least one common label (e.g. rdfs:label), and stores them as aligned results. Please note that it deletes non-English and non-Numeric characters from the labels of input entities and transforms the labels into lowercase characters before it starts to match entities.

IDMatcher The matching procedure of this module is the same as that of LabelMatcher, except that it finds a best matched entity in the second input ontology for each entity in the first input ontology that has identical ID (e.g. rdf:ID).

LCSMatcher It finds a best matched entity with highest LCS [5] (Longest Common Subsequence) similarity in the second ontology for each entity in the first input ontology and stores them as aligned results. When it computes the LCS similarity of

two input entities, it first delete non-English and non-Numeric characters from all labels (e.g. rdf:ID, rdfs:label, rdfs:comment) of the input entities. Then it computes the LCS similarities of each pair of labels between the input entities and considers the highest similarity as the final similarity of the input two entities. The LCS similarity of two input labels can be computed using the following equation:

$$LCS\ Similarity(A, B) = \frac{2 \times LCSlen(A, B)}{Length(A) + Length(B)}$$

In above equation, A and B mean the input labels, function LCSlen(A,B) returns the length of longest common subsequence between A and B, and functions Length(A) and Length(B) returns the lengths of A and B respectively.

SMOAMatcher The matching procedure of this module is the same as that of LCSMatcher, except that it replaces the LCS similarity computing scheme with the SMOA [4] similarity computing scheme.

PurityMatcher The matching procedure of this module is similar to that of LabelMatcher and IDMatcher, except that it deletes all useless English stopwords (such as words "has") of all labels within the classes and properties in the input ontologies before it starts to match ontologies. It can find interesting aligned results such as the mapping of labels "has_an_Email" versus "email".

TFIDFMatcher This module matches only classes from different input ontologies based on the TF-IDF [1] Cosine similarity [2] computing scheme. The idea of exploiting text-mining techniques (such as TF-IDF representation) in the system is inspired by YAM++ version 2012 [6]. The matching procedure of this module is described as follows. For each class in the first input ontology, it computes the TF-IDF Cosine similarities of the class and all classes in the second ontology. Then it chooses the best matched class with highest similarity in the second ontology, and stores them as aligned results. When it tries to compute the TF-IDF Cosine similarity of two input classes, it first splits the all labels (e.g. rdf:ID and rdfs:label) of input classes into two English token sets, and then it computes the TF-IDF values of each token within the two token sets respectively. Please note that the TF value of a token means the frequency of this token appears in the token set, and the IDF value of a token means the inverted frequency of this token appears in all token sets that all classes hold in the ontology. After that, it normalizes the TF-IDF values of two token sets, considers them as two normalized TF-IDF vectors, and finally computes the Cosine similarity of these two TF-IDF vectors.

NETMatcher It finds a best matched class with highest NET (named-entity transformation) similarity in the second ontology for each class in the first input ontology and stores them as aligned results. When it tries to compute the NET similarity of two input classes, it first deletes non-English and non-Numeric characters of all labels (e.g. rdf:ID and rdfs:label) of input classes and splits them into tokens. Please note that if there are n tokens and n is no less than 2, then at least n-1 tokens leads by capital English character or numeric character. Then it computes the input classes' NET similarity using the following equation:

$$NETSimilarity(A, B) = \frac{commonPrefix((A \cup B) - (A \cap B)) + commonTokens(A \cap B)}{n}$$

In above equation, A and B mean the token sets belong to different input classes, function commonTokens returns the total common tokens of input token sets, function commonPrefix returns the average of total common prefix characters versus total characters of all tokens within different input token sets. This module can find interesting aligned results such as the mappings of tokens "OWL" versus "Web Ontology Language" or "PCMembers" versus "Program Community Members", etc.

1.2.2. Structural Level Modules

There are two structural level matching modules, PBCTMatcher and PBCSMatcher in the system now. The former computes classes' integrated similarities using token-based computing scheme and the latter computes them using string-based ones. The ideas of the above matching modules are derived from the matcher NameAndPropertyAlignment of Alignment API 4.5 [3].

PBCTMatcher The full name of it is Property-based Class Token Matcher. For each class in the first input ontology, it finds a best matched class with highest integrated similarity in the second input ontology. It computes input classes' integrated similarities by combining the input classes' similarities and their properties' similarities using the following equation:

$$Integrated\ Similarity = \begin{cases} 0.5 \times CS_{tfidf} + 0.5 \times PS_{tfidf} & if\ PS \geq 0.5 \\ 0 & if\ PS < 0.5 \end{cases}$$

In the above equation, CS_{tfidf} means the TF-IDF Cosine similarity between the input classes, and PS_{tfidf} means the TF-IDF Cosine similarity between the belonged properties of input classes. The computing procedure of TF-IDF Cosine similarity is the same as that of TFIDFMatcher.

PBCSMatcher The full name of it is Property-based Class String Matcher. It's like PBCTMatcher, except it computes input classes' integrated similarities using LCS (Longest Common Subsequence) similarity computing scheme rather than using TF-IDF similarity computing scheme in PBCTMatcher.

1.2.3. Optimization Level Modules

ThresholdFilter It filters the stored aligned results in each matching module according to the default filter threshold, respectively. Each aligned result whose similarity is lower than the specified filter threshold is deleted from the original matching module.

AlignmentMerger It merges all stored aligned results of each matching module by a special order. The merging type of AlignmentMerger is called Absorb. That means when it merges the aligned results of two matching modules, it preserves all aligned results of the former and filters any aligned results of the latter which is partly or

completely overlapped in the former. A merging example of AlignmentMerger is given in Fig. 2.

Fig. 2. A merging example of AlignmentMerger.

In Fig. 2, AlignmentMerger is to merge the aligned results of the matching modules A_1 and A_2. Let $C_{i,j}$ be the j_{th} object in ontology i. If the aligned results in A_1 are { $<C_{1,1},C_{2,1}>$, $<C_{1,2},C_{2,4}>$ } and the ones in A_2 are { $<C_{1,2},C_{2,5}>$, $<C_{1,3},C_{2,8}>$ }. Because $<C_{1,2},C_{2,5}>$ in A_2 is partly overlapped with $<C_{1,2},C_{2,4}>$ in A_1, the merged aligned results are thus { $<C_{1,1},C_{2,1}>$, $<C_{1,2},C_{2,4}>$, $<C_{1,3},C_{2,8}>$ }.

1.3 Adaptations made for the evaluation

ODGOMS uses the same parameters to run each experiment in all tracks of OAEI 2013. The parameters are divided into two groups as follows.

The first group of parameters includes the default filter thresholds used by module ThresholdFilter in the system, which are set to be 1.0 for modules LabelMatcher, IDMatcher, and SMOAMatcher, 0.87 for modules LCSMatcher, PurityMatcher, and NETMatcher, 0.8 for module PBCSMatcher, 0.781 for module TFIDFMatcher, and 0.3 for module PBCTMatcher, respectively.

The second group of parameters includes the merging order used by module AlignmentMerger in the system. The mentioned above merging order is : LabelMatcher, IDMatcher, LCSMatcher, SMOAMatcher, PurityMatcher, TFIDFMatcher, NETMatcher, PBCTMatcher, and PBCSMatcher.

1.4 Link to the system and parameters file

The readers can download execution files of all versions of ODGOMS from our Google SkyDrive download position[1], and test them using SEALS client 4.1. Please refer to SELAS client tutorial[2] to learn more testing examples.

2 Results

In this section, the OAEI 2013 official results of ODGOMS are listed in from Table 1 to Table 5, and they can be find on the OAEI 2013 website too.

Since some tasks in Largebio track are time-consuming and ODGOMS cannot finish those tasks in 18 hours, we have run ODGOMS for three lightweight tasks of Largebio track by SEALS client 4.1 at local side, and have listed the results in Table 6. The mentioned above experiments are executed on a PC with Intel Core i7-3770S CPU (3.10GHz), 4GB RAM, and Ubuntu 12.04 LTS (64-bit version).

1ODGOMS download position: http://goo.gl/SKkhnU
2http://oaei.ontologymatching.org/2013/seals-eval.html#tutorial

2.1 Benchmark

The official results of ODGOMS version 1.2 released from OAEI 2013 website are listed in Table 1.

Test Datasets	Precision	Recall	F-Measure	Execution Time(s)
Benchmark – biblioc	**0.99**	**0.55**	**0.71**	**100**
Benchmark – biblioc (weighted)	**0.98**	**0.54**	**0.70**	**100**

Table 1. The results for Benchmark track.

2.2 Anatomy

The official results of ODGOMS version 1.2 released from OAEI 2013 website are listed in Table 2.

Test Datasets	Precision	Recall	F-Measure	Execution Time(s)
Anatomy Testsuite	**0.979**	**0.712**	**0.824**	**1,212**

Table 2. The results for Anatomy track.

2.3 Conference

The official results of ODGOMS version 1.2 released from OAEI 2013 website are listed in Table 3. In Table 3, the pre-test results (ra1) are listed in the first row, and the blind-test results (ra2) are listed in the second row.

Test Datasets	Precision	Recall	F-Measure	Execution Time(s)
Conference Testsuite (ra1)	**0.74**	**0.60**	**0.66**	**19**
Conference Testsuite (ra2)	**0.7**	**0.55**	**0.62**	**19**

Table 3. The results for Conference track.

2.4 Multifarm

The official results of ODGOMS version 1.2 released from OAEI 2013 website are listed in Table 4.

Test Datasets	Precision	Recall	F-Measure	Execution Time(s)
Multifarm – Different ontologies (i)	**0.26**	**0.06**	**0.10**	**2,640**
Multifarm – Same ontologies (ii)	**0.47**	**0.03**	**0.05**	

Table 4. The results for Multifarm track.

The results show that the F-Measures of the MultiFarm track are not good. We think the reasons for these results are that ODGOMS is not designed to match ontologies which are written in completely different languages yet.

2.5 Library

The official results of ODGOMS version 1.1 (not version 1.2 in this track) released from OAEI 2013 website are listed in Table 5. In this track, ODGOMS got the highest F-measures of all attended systems. By the way, in our local test the results of ODGOMS version 1.2 is slightly better than it of version 1.1.

Test Datasets	Precision	Recall	F-Measure	Execution Time(s)
Library Testsuite	**0.698**	**0.830**	**0.758**	**27,936**

Table 5. The results for Library track.

2.6 Largebio

We run ODGOMS for three small tasks of Largebio track by SEALS client 4.1 at local side. The results are listed in Table 6. In Table 6, the F-Measures are identical to the official results released on OAEI 2013 website except the execution time of the former are faster than the latter. The results of SNOMED-NCI (small) are not shown in the official results on OAEI 2013 website since its execution time exceeded the maximum limit of 18 hours.

Test Datasets	Precision	Recall	F-Measure	Execution Time(s)
FMA-NCI (small)	**0.953**	**0.831**	**0.888**	**6,578**
FMA-SNOMED (small)	**0.862**	**0.570**	**0.686**	**30,046**
SNOMED-NCI (small)	**0.873**	**0.622**	**0.726**	**245,434**

Table 6. The results for Largebio track.

3 General Comments

3.1 Comments on the results

The official results of OAEI 2013 show that ODGOMS is competitive with other well known ontology matching systems in all OAEI tracks, especially in Library track it got the highest F-measures of all attended systems. The worst performance is happened in Multifarm track. The reason is that ODGOMS is not designed to deal with purely multilingual ontology matching problems yet.

3.2 Discussions on the way to improve the proposed system

ODGOMS exploits simple string-matching schemes and text-mining techniques to match ontologies now. It suffers from the following two problems. The first one is that it cannot optimize the results for each matching question automatically. The second one is that it cannot perfectly deal with purely multilingual ontology matching problems.

In order to solve the above two problems, we are extending the new abilities into the system as follows. For the first problem, we will apply machine learning technologies into the system so that it can find the best parameters that can be used in the system automatically when it deals with different ontology matching questions. And for the second problem, we will add the off-line translation ability between foreign languages and English into the system so that it doesn't need the help of on-line translation API (e.g. Microsoft On-Line Translation API).

4 Conclusion

It's the first time ODGOMS attended OAEI campaign. Although it got good results in almost all OAEI 2013 tracks, but it still suffers from some problems such as time-consuming and multilingual problems. The further research topics would be extend the machine learning and multilingual abilities into the system. We hope the performance of it can be improved when it attends the OAEI campaign next year.

References

1. Gerard Salton and Chris Buckley. Term Weighting Approaches in Automatic Text Retrieval. Information Processing and Management, 24 (5): 513-523, 1988.
2. Amit Singhal. Modern Information Retrieval: A Brief Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 24 (4): 35-43, 2001.
3. Jerome David, Jerome Euzenat, Francois Scharffe, and Cassia Trojahn dos Santos. The Alignment API 4.0. Semantic Web Journal, 2 (1): 3-10, 2011.
4. Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. A String Metric for Ontology Alignment. Lecture Notes in Computer Science (LNCS), 3729: 624-637, 2005.
5. L. Bergroth, H. Hakonen, and T. Raita. A Survey of Longest Common Subsequence Algorithms. SPIRE (IEEE Computer Society) 00: 39-48, 2000.
6. DuyHoa Ngo and Zohra Bellahsene. YAM++ - Results for OAEI 2012. In Seventh International Workshop on Ontology Matching (OM 2012): 226-233, 2012.