Overview of YAM++-(not) Yet Another Matcher for ontology alignment task

Duy Hoa Ngo, Zohra Bellahsene

▶ To cite this version:

Duy Hoa Ngo, Zohra Bellahsene. Overview of YAM++-(not) Yet Another Matcher for ontology alignment task. Journal of Web Semantics, 2016, 41, pp.30-49. 10.1016/j.websem.2016.09.002 . hal-01411159

HAL Id: hal-01411159
https://hal.science/hal-01411159

Submitted on 14 Jun 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Overview of YAM++ - (not) Yet Another Matcher for ontology alignment task

DuyHoa Ngo, Zohra Bellahsene

University Montpellier, LIRMM
161 rue Ada, 34095, Montpellier, France

Abstract

Several challenges to the field of ontology matching have been outlined in recent research. The selection of the appropriate similarity measures as well as the configuration tuning of their combination are known as fundamental issues the community should deal with. Verifying the semantic coherence of the discovered alignment is also known as a crucial task. As the challenging issues are both in basic matching techniques and in their combination, our approach is aimed to provide improvement at the basic matcher level and also at the level of framework. Matching large scale ontologies is currently one of the most challenging issues in ontology matching field. The main reason is that large ontologies are highly heterogeneous both at terminological and conceptual levels. Furthermore, matching very large ontologies entails exploring a very large searching space to discover correspondences. It may also require a huge amount of main memory to maintain the temporary results at each computational step. These factors strongly impact the effectiveness and efficiency of any ontology matching tool. To overcome these issues, we have developed a disk-based ontology matching approach. The underlying idea of our approach is that the complexity and therefore the cost of the matching algorithms are reduced thanks to the indexing data structures by avoiding exhaustive pair-wise comparisons. Indeed, we extensively used indexing techniques in many places. For example, we defined a bitmap encoding the structural information of an ontology. This indexing structure will be exploited for accelerating similarity propagation. Moreover, our approach uses a disk-based mechanism to store temporary data. This allows to perform any ontology matching task on a simple PC or laptop instead of a powerful server.In this paper, we describe YAM++, an ontology matching tool, aimed at solving these issues. We evaluated the efficiency of YAM++ in various OAEI 2012 and OAEI 2013 tracks. YAM++ was one of the best ontology matching systems in terms of F-measure. Most notably, the current version of YAM++ has passed all scalability and large scale ontology matching tests and obtained high matching quality results.

Keywords: Ontology Matching, Similarity Measure, Matcher Combination, Similarity Propagation, Mapping Selection, Large Scale Ontology Matching.

1. Introduction

In recent years, ontologies have attracted a lot of attention in Computer Science, especially in the Semantic Web field. They serve as explicit conceptual knowledge models and provide the semantic vocabulary that make domain knowledge available to be exchanged and interpreted among information systems. Hence, they open new opportunities for developing a new line of semantic applications such as semantic search [28, 51], semantic portal [82, 50, 44], semantic information integration [12, 71, 3], intelligent advisory systems [74, 4], semantic middleware [43, 7], semantic software engineering [13], etc. However, one of the most difficult issues is how to deal with heterogeneity of ontologies [41, 29]. Due to the decentralized nature of the semantic web, an explosion in the number of ontologies is expected. Many of them may describe similar domains, but they are very different because they have been designed and developed independently by different ontology engineers following diverse modeling principles and patterns.

For example, within a collection of ontologies describing the domain of organizing conferences [21]. People attending to the conference can be conceptualized with different names such as `conference_Participant`, `attendee`, `participant`, `delegate`, `listener`[1]. The heterogeneity of ontologies mainly causes problems of variation in meaning or ambiguity in entity interpretation and, consequently, it prevents information systems from sharing their own domain knowledge to the community. Therefore, without knowing the semantic mappings between entities of ontologies, information systems cannot perform interaction, communication and collaboration with each other.

According to [24], ontology matching is a key solution to the semantic heterogeneity problem. It discovers correspondences between semantically related entities of ontologies. Ontology matching can be done either by hand or by using (semi) automatic tools. Discovering manually mappings is tedious, error-prone, and imprac-

Email addresses: `elendh@nus.edu.sg` (DuyHoa Ngo), `bella@lirmm.fr` (Zohra Bellahsene)

[1](in conference dataset: confOf.owl, ekaw.owl, edas.owl, iasted.owl, sigkdd.owl)

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

tical due to the number, size and heterogeneity of ontologies. Hence, the development of fully or semi automatic ontology matching tools becomes crucial to the success of the semantic information systems and applications. In the last decade, through the annual campaign OAEI[2], many ontology matching systems/tools have been proposed. These state-of-the-art approaches have made a significant progress in the ontology matching field, but none of them gained a clear success in terms of matching quality for all the matching scenarios [23]. In [25, 79, 73, 52], challenging issues in ontology matching have been described in detail. Among these challenges, selecting the appropriate similarity measures as well as tuning the configuration of their combination are the toughest fundamental problems of all matching systems. Matching scenarios may require to combine the outcome of the used similarity measures in a different way. Furthermore, the difficulty of the problem grows with the size of the ontologies. Indeed, matching large scale ontologies is one of the most difficult problems in ontology matching field. In particular, the size of ontologies being matched strongly impacts the performance, i.e., effectiveness and efficiency of any ontology matching system. The main reasons are: (i) large ontologies usually lead to a high conceptual heterogeneity and (ii) The complexity of matching is usually proportional to the size of the input ontologies. Furthermore, discovery mappings in a huge space is very time consuming especially if multiple matchers need to be evaluated and combined. Thus, the efficiency of the matching system will be degraded.

To deal with large-scale ontology matching, several techniques have been proposed. The most promising approaches are: filtering-based methods, partitioning methods and background-based ones. The main idea behind these techniques of filtering methods is to reduce the search space by heuristically eliminating less promising candidate mappings. For example, in Eff2Match [8], the heuristic to select candidate mappings for each entity in the source ontology is taken by performing the top-K entities algorithm in the target ontology according to their context (Virtual Document) similarity. More sophisticated heuristics strategies based on different extracted features such as label, hierarchy, neighbors, etc. are applied in each iteration to select the promising mappings [16], ServOMap [14].

While in partitioning-based methods, two large ontologies are firstly divided into sub-ontologies according to their structural information. Then the alignment process is performed between entities of pairs of sub-ontologies. In order to avoid exhaustive pair-wise comparisons, only the high relevant pairs of sub-ontologies will be passed to the matching process. These methods can be found in Falcon-AO [33] and COMA++ [2].

A sub-class of this category is known as anchor-based partitioning methods. These methods are a modified version of the algorithms above, which partition to-be-matched ontologies are done according to the set of anchors. In short, an anchor is a pair of entities mapping determined by a similarity measure. A fragment or sub-ontology is constructed by collecting neighbors entities of the chosen anchors. Then, the alignment process will be performed for each pair of related sub-ontologies. These methods can be found in Anchor-Prompt [70], AnchorFlood [31], Lily [84], TaxoMap [30].

The underlying idea of our approach is that the annotation, the structural and the contextual information of entities are indexed in order to improve the whole matching process both in terms of matching quality and time performance. Unlike those of related work, our filtering methods make use of the annotation-based indexes in order to accelerate the filtering process. Furthermore, the structural indexes are also exploited to check the coherence of the resulting mappings. Indeed, in addition, verifying the semantic coherence of the discovered alignment is known as a challenging issue in large scale ontology matching because almost all reasoning systems fail or cannot completely classify large ontologies. We have implemented a new inconsistency removing algorithm based on Clarkson algorithm for the weighted minimum vertex cover problem. The details of this contribution can be found in [62]. In this paper, we highlight the main contributions and techniques that have been implemented in YAM++ and that have made it one of the best of ontology matching tools. These contributions are the following:

- Effective and efficient filtering methods to deal with large scale ontology matching.
- A heuristic-based label similarity measure which integrates a strict heuristic filter with the label similarity measure, which is aimed at detecting of *informative words*.
- A machine learning-based method to combine terminological similarity measures without the effort of manual setting.
- An information retrieval-based similarity measure to improve the matching quality and to deal with terminological heterogeneity. This new similarity measure takes into account not only syntactic similarity but also information content of words. This measure constitutes an alternative to machine learning method when training data are not available or in large scale setting.
- A bitmap encoding the structural information of an ontology that is exploited for accelerating the similarity propagation. This method is stable and reliable because it exploits and uses all the structural information of an ontology for discovering mappings.
- A dynamic weighted sum method to combine the mappings resulting from the element matcher and structure matcher. The benefit is that it automatically assigns weights to each matcher for a given matching scenario. Moreover, it also automatically determines the filter's threshold value to produce the final mappings.
- A fast semantic filtering method to detect the inconsistent mappings when matching large ontologies.
- The experimental results demonstrate that YAM++ is both effective in terms of quality of the alignments, and efficient in terms of time performance and scalability.

YAM++ was one of the very best system in OAEI competitions from 2011 to 2013. Particularly, thanks to

[2]http://oaei.ontologymatching.org/

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

the contributions on dealing with terminological heterogeneity (i.e., machine learning, information retrieval methods), structural heterogeneity (i.e., propagation method), YAM++ achieved the best results in the series of `Systematic Benchmark` tracks in years 2012 and 2013; in the `Conference` track in years 2011, 2011.5, 2012 and 2013; especially, in the *multilingual* `Multifarm` tracks in years 2011.5, 2012 and 2013. Additionally, thanks to the fast semantic indexing and the inconsistency filtering method that has been devoted to large scale ontologies matching, YAM++ was one of the best systems on the `Anatomy` track, `Library` track and `Large Biomedical Ontologies` tracks in years 2012 and 2013.

The rest of the paper is organized as follows: Section 2 provides the basic notions and definitions used in this paper as well as the evolution and the architecture of YAM++. In Section 3, we present our method for extracting and indexing the annotation, the structural and the contextual information of entities within an ontology in order to improve the whole matching process.Then, we present our approach for candidates filtering in Section 4. Section 5 describes our approach to deal with the problem of terminological heterogeneity. While Section 6 is devoted to structural heterogeneity. In Section 7, we will present our method to combine the mapping results obtained from the element level with the ones obtained from the structure level. Section 8 provides an overview of our approach for semantic coherence verification. The experiments and results over OAEI2012 datasets are described in Section 9. Next, Section 10 contains an overview of related work. Finally, the conclusion and future work are discussed in Section 11.

2. Overview of our approach

2.1. Background

Firstly, we present the ontology matching problem and the main related notions used in this paper.

Ontology matching is the process of discovering correspondences (mappings) between entities belonging to different ontologies. Whereas, a **correspondence** or a **mapping** m is defined as a four-tuple of the form: $m = \langle e_s, e_t, r, k\rangle$ where: e_s and e_t are entities in source O_s and target O_t ontologies respectively, r is a relation (e.g., equivalent, subsume) and $k \in [0,1]$ represents the degree of confidence of this relation[24].

An **alignment** is a set of correspondences between two or more (in case of multiple matching) ontologies.

A **matcher** is a matching algorithm, which is aimed to discover correspondences between ontologies.

An ontological **entity** can be a class, a property or a data instance. It is identified by an Internationalized Resource Identifiers(IRI) and may be annotated by some human reading labels. In YAM++, the suffix identifier and annotated labels are commonly called entity's **labels**. Additionally, YAM++ is an ontology matching tool on schema level, therefore, it only focuses on discovering correspondences between classes or between properties.

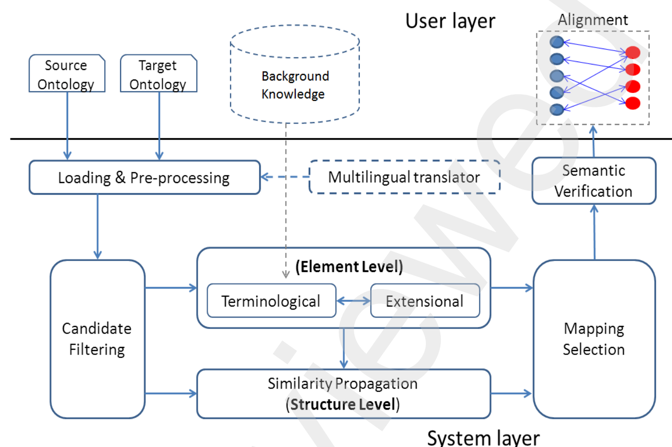


Figure 1: General Architecture of YAM++

2.2. YAM++ architecture

The evolution of YAM++ can be seen through its publications since its first time participation to the OAEI 2011 campaign. However, the previous publications either describe a specific technique (e.g., machine learning approach in [69], information retrieval method in [66], structural method in [68]) or simply focus on experimental evaluation in OAEI competition (e.g., [65, 60, 61]). This paper is compiling previous contributions, as well as some extensions in detail to provide a whole picture of YAM++ and the techniques that have been implemented. YAM++ was one of the very best system in OAEI competitions from 2011 to 2013.

Those contributions are reflected through the main components of YAM++ as shown in Figure 1. The input to YAM++ is **source** and **target** ontologies, which are encoded in RDFS[3] or OWL[4] languages. Optionally, **background knowledge** such as the lexical resource WordNet[5] or training data for the future use in machine learning method can be provided by the user.

Having two ontologies as the input, YAM++ works as follows. First, the input ontologies are loaded into the main memory within the **loading & pre-processing** component. For each input ontology, the annotation and structure information of all the entities are extracted. In addition, if the annotations are not in English, a **multilingual translator** (Microsoft Bing Translator API[6]) is used to translate the annotation into English. Next, the **candidate filtering** module aims to reduce the search space. In particular, it applies some heuristic filter to select the most potential candidate mappings (see Section sec:Filtering). The similarity of candidate mappings is computed at the two consecutive levels, i.e., element and structure. The aim of the **element level** module is to discover as many as possible high accuracy mappings by analyzing elements in isolation and ignoring (at this stage) their relations with others. This module consists of two sub-modules: **terminological** matcher and **extensional**

[3]http://www.w3.org/TR/rdf-schema/
[4]http://www.w3.org/TR/owl2-syntax/
[5]http://WordNet.princeton.edu/
[6]http://www.bing.com/translator

Preprint not peer reviewed
This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

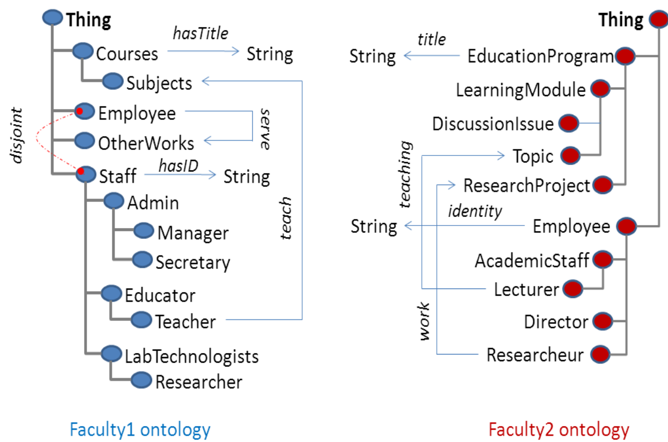


Figure 2: Motivating example

matcher. The Terminological matcher exploits the annotation information of entities to compute similarity scores, whereas, the Extensional matcher exploits the information from the data instances accompanying the ontologies. The mappings discovered by the **element level** matcher are passed to the **structure level** matcher in order to discover new mappings by analyzing the position of the entities in the taxonomy of ontologies. The main intuition of this part is that if two entities are similar, their neighbors (in the same relations) might also be somehow similar. In our current system, we have implemented a popular graph-based technique to deal with ontologies - **similarity propagation** in the structure level matcher module (see Section 6 for more details). Next, the outputs of the element level and structure level matchers are passed to the **mapping selection** module, which will be described in Section 7. Then, the discovered mappings are passed to the **semantic verification** in order to detect and remove the inconsistent ones.

For the purpose of illustration, we propose a simple example of matching scenario which involves *two faculty ontologies* shown in Figure 2. Here, the classes in the two ontologies are represented by ovals in different colors. The properties and relationships of classes are showed as labeled arrows. Non-equivalent relations (e.g., subsumption) can be derived from the discovered equivalent relations through description logic reasoners with external resources, such as WordNet, domain ontologies or text corpora. It is another challenge in ontology matching task thus requires different techniques using in equivalent relations discovery [32, 26, 81]. In the scope of our research, YAM++ focuses in discovering only equivalent mappings between entities (i.e., classes and properties).

In order to cope with large scale ontology matching, further adaptations have been applied. Particularly, inside the **loading & pre-processing** component, YAM++ indexes ontological information of concepts such as annotation, structure and context for fast and efficiently access the needed information. In order to save the working memory (RAM) for large scale ontology matching, all three indexing folders, i.e., **Annotation Indexing**, **Structure Indexing** and **Context Indexing** are stored in disk and located in operating file system. The indexes will be loaded from disk to main memory when they are needed for some steps in the algorithm and they will be released from main memory as they have done their jobs. Thus, we call the current YAM++ version a **disk-based** ontology matching system. Additionally, due to high computational cost in detection of inconsistency in the discovered alignment between large scale ontologies, in the **semantic verification**, instead of using powerful ontological reasoner, YAM++ has been adopted a a greedy heuristic strategy accompanied with a bank of conflict patterns to refine the resulted alignment. Finally, it is worthy to noticeable that YAM++ is able to work in small and large scale ontology according to the size of the input ontologies. Indeed, If the size is higher than 1000, it switches to large scale regime.

3. Indexing Ontologies

In this section, we present the way that the annotation, the structural and the contextual information of entities within an ontology will be extracted and indexed in order to improve the whole matching process. Here, the annotation information reflects the textual description of an individual entity for human reading. While the structural information shows the semantic relationships between an entity and the other entities in the ontology. Finally, the contextual information reflects the textual context that is closely related to the entity's terms.

3.1. Annotation Indexing

The aim of annotation indexing is to filter candidate mappings by alike labels of entities and to compute similarity scores between them. In our approach, the first aim can be quickly achieved by label and sub-label indexing; whereas, the second one, which requires the weight value of each term appearing in the labels [67], can be done by term indexing. Let us illustrate the indexing procedure through the following example "`http://human.owl#NCI_C32696`" as shown in Fig. 3.

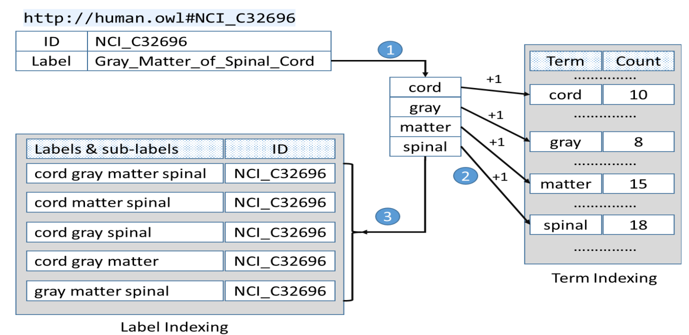


Figure 3: Example of annotation indexing

This concept has a label "`Gray_Matter_of_Spinal_Cord`". At the step 1, the normalization of a label consists in tokenizing it into tokens, removing stop-words, stemming words and sorting them in alphabet order. It guarantees that there is a unique normalization of a given label. As shown in Fig. 3, the normalization produces 4 terms such as "`cord`", "`gray`", "`matter`" and "`spinal`". At the step 2,

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

those terms are stored in a table with their number of occurrences. Each time a term appears, its counter increases by 1. Once all entities have been indexed, the information content of a word t appearing in a concept label of an ontology is computed in the following manner.

$$IC(t) = \log \frac{|T|}{|N|}, \quad (1)$$

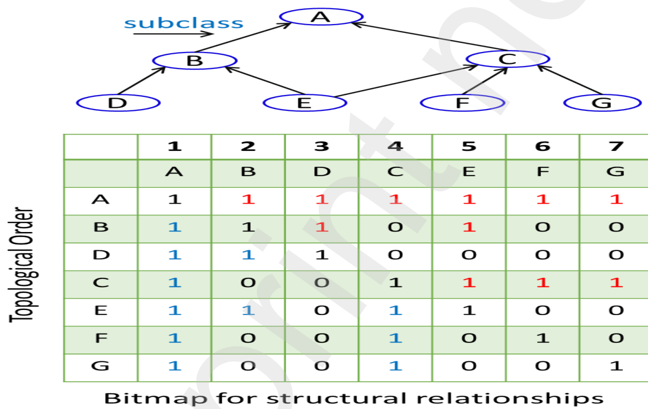
where, $|T|$ is the total number of concepts in a given ontology and $|N|$ is the number of labels containing the word t. On this basis, a word is assigned a weight as follows:

$$weight(t) = \frac{IC(t)}{\max_{i=1..|T|}\{IC(t_i)\}}. \quad (2)$$

Next, the term's weight will replace its counter value in the index table. It will be used in the similarity calculation between two labels of entities. At the step 3, we use an inverted map to index labels and its sub-labels of all the entities. Here, the key column includes unique labels and sub-labels returned by the normalization process. The value column is the ID of entities. In our approach, a sub-label is generated by removing one word from the normalized label. As it is shown in Fig. 3, a normalized label and 4 sub-labels have been indexed for the concept with ID "`NCI_C32696`".

3.2. Structure Indexing

In order to quickly access to the semantic information of a given concept in the ontology, indexing ontological structure is necessary. Each concept is assigned a topological order and is encoded by a bitmap containing all information of its ancestors and descendants. Let's illustrate the indexing process through a small ontology fragment as shown in Fig. 4. This indexing structure is used in two places: (i) in similarty propagation process and (ii) in semantic verification.



	1	2	3	4	5	6	7
	A	B	D	C	E	F	G
A	1	1	1	1	1	1	1
B	1	1	1	0	1	0	0
D	1	1	1	0	0	0	0
C	1	0	0	1	1	1	1
E	1	1	0	1	1	0	0
F	1	0	0	1	0	1	0
G	1	0	0	1	0	0	1

Figure 4: Example of a bitmap encoding the structural information

Firstly, a topological sort is performed over all classes of the given ontology. The topological order is {A, B, D, C, E, F, G } shown in the first column of the matrix in Fig. 4. For example, because the topological order of class C is 4 we first set value 1 to the *4th* bit of a bitmap length 7. Next, by performing an upward transition through `SuperClass` relation and downward transition through `SubClass` relation for all concepts we obtain the full matrix as shown in Fig. 4. Here, for a given class, its ancestors are marked by bits **1** on the left side, whereas, its descendants are bits **1** on the right side of its indexing position.

It is noticeable that the bitmaps assigned to classes use large storage space thus downgrade computational performance. Moreover, they are very sparse due to the ancestors and descendants of a class constitutes only a small subset of the whole classes in the ontology. Therefore, a compressed bitmap is needed for saving space and improving computational performance. In YAM++, we have applied a compression technique based on the *Word Aligned Hybrid* bitmap compression algorithm, which is the currently recognized as the most efficient one, mainly from a computational perspective [85]. The detail of this technique is described in [10].

Thanks to the bitmap encoding table, the structural relations of classes can be easily and quickly accessed. For example, finding the lowest common ancestor (LCA) of any two classes can be done by running logical *AND* operator over their ancestors that will return the class whose topological order corresponds to the index of the last bit 1 in the result. In this example above, $ancestors(C) =$ **1001000** and $ancestors(D) =$ **1110000**.
Because **1001000** & **1110000** = **1000000** has the right most bit 1 at the first position, thus, the lowest common ancestor of classes C and D is class A.

On the other hand, the disjoint relation of any two classes can be easily checked as follows:

$$disjoint(X,Y) \leftarrow \exists(A,B) \mid A \in ancestors(X) \vee B \in ancestors(Y) \ \vee disjoint(A,B)$$

Similarly, to list all pairs of disjoint classes in an ontology, the following rule can be applied:

$$disjoint(X,Y) \rightarrow disjoint(A,B) \mid \forall A \in descendants(X), \ \forall B \in descendant(Y)$$

In our approach, disjoint classes of a given class are also encoded in a bitmap, in which positions of bit 1 indicate their topological order. All the bitmap encoding subclass, superclass and disjoint information are stored in a structure index folder. Based on the structure-based index $strucInd$ of a given ontology, the following supporting functions have been defined: $GetAncestors(c, strucInd)$, $GetDescendants(c, strucInd)$, $GetDisjoints(c, strucInd)$ returns all ancestors, descendants (including c) and concepts that are disjoint with the concept c respectively.

3.3. Context Indexing

The aim of the context indexing is to provide a fast and reliable search method to discover similar concepts among various ontologies by exploiting the concepts' description. Our heuristic is that "The vocabularies describing the context of alike concepts in the same domain are highly similar". In our approach, for each ontology, a context index is built as a vector space model (VSP), which is an algebraic model for representing text documents as vectors of index terms. Here, contextual information of a concept is represented through 3 documents such as:

Preprint not peer reviewed
This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

- *Individual document* is a string consisting of annotations (i.e., labels, synonyms, comments of a given concept).
- *Ancestor document* is a string consisting of all individual documents of the concept's ancestor.
- *Descendant document* is a string consisting of all individual documents of the concept's descendants.

Therefore, in our vector space model, each concept' ID is mapped to 3 fields corresponding to its 3 types of contextual documents. Inside the vector space model, terms are tokenized from documents, then indexed and assigned with a real value determined by the `TF-IDF` weighting scheme. Consequently, each document is transformed into a vector of real values, thus enables to calculate similarity scores between concepts.

4. Candidates filtering

The candidate filtering method is a very important step for scalability. It contributes to reduce the search space and consequently to decrease computational time by eliminating the potential candidate mappings having a low similarity score. In general, the filtering method simply lists all candidates mapping, in which both of elements belong to the same ontological type such as concept vs. concept, object property vs. object property, data property vs. data property and data individual vs data individual.

In large scale matching task, YAM++ filters only candidate mapping whose elements have highly similar labels. Instead of implementing any string similarity measure, YAM++ makes use of Hash function, which is a built-in function of hashmap and hashset data structure. The principle is, the same strings will return the same hash values. Therefore, we can easily determine the overlapped key set from the two label indexes of the source and target input ontologies. From the common key set, we can filter out a collection of potential candidate mappings. The output of this process is a *filtering candidate two-column table*, in each of its row, the first column contains the ID of a concept of the source ontology; whereas, the second column contains a list of tuples consisting of the ID of a concept of the target ontology and a score value indicating how much the source and target concepts are similar. Let us illustrate the filtering candidates for *mouse*, i.e., source ontology and *human*, i.e., target ontology in Fig. 5.

Here, both the source and target label index have a common key such as "cord matter spinal", thus each pair of concepts' ID (i.e., `MA_0000002` and `NCI_C32696`) corresponding to those keys produces a candidate mapping. Additionally, we assign a score value to each candidate mapping as follows.

$$score = \mathbf{LabelSim}(label_s, label_t) \times \mathbf{w}(label_s) \times \mathbf{w}(label_t) \quad (3)$$

where $label_s$ and $label_t$ are labels of the *source* and *target* concepts respectively; function *LabelSim* is the label-based similarity measure described in Section 5.2; the weight

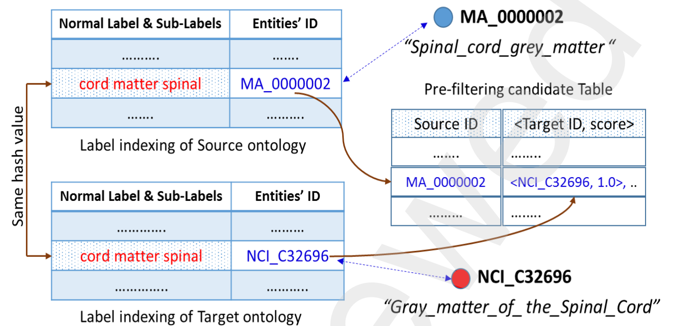


Figure 5: An example of filtering by labels and sublabels

function w of a $label_i$ among N labels described in concept C is determined such as:

$$w(label_i) = c_{type} \times \frac{|norm(label_i)|}{|\cup_{j=1:N} norm(label_j)|} \quad (4)$$

Here, $|S|$ represents the size of S; $norm(label)$ returns a set of unique terms in the *label* after its normalization (i.e., tokenizing, stop-word removing and stemming) by *norm* function. The $\cup$ function is a set union operator. The coefficient c_{type} value is defined according to the position of a label described in the concept as follows.

$$c_{type} = \begin{cases} 1.0 & when \text{ type } is \text{ ID label} \\ 0.9 & when \text{ type } is \text{ synonym} \end{cases}$$

The idea of assigning a weight to each label is important due to the fact that the more different labels a concept has, the more ambiguous each of its labels is. Consequently, they have a lower weight. If a concept has only one label, this label's weight will be equal to 1.0. Moreover, we distinguish the meaning level of different types of labels by assigning different coefficient c_{type}. Our heuristic is that the similarity of two concepts' labels is more potential than similarity of their synonym labels. In the example above, both concepts `MA_0000002` and `NCI_C32696` have only one label, the weight values of the both labels are 1.0. The similarity value calculated by the two labels of those concepts is equal to 1.0 (see example in Section 5.2). Therefore, the final score assigned to this candidate mapping is 1.0.

5. Dealing with terminological heterogeneity at element level

In this section, we present two approaches dealing with terminological heterogeneity to discover entities having similar labels. Firstly, in YAM++ version 2011, a machine learning method is designed upon available training data to combine different similarity measures. However, its performance strongly depends on the training data, i.e., if the training data are not available or not suitable, the matching quality is poor. Therefore, a new information content-based similarity measure was proposed to replace the ML method in YAM++ version 2012 and later.

5.1. Combining terminological similarity measures

The high heterogeneity of ontologies leads to different types of mismatches between their entities [80] therefore no single similarity measure can be efficient in all the

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

cases. For instance, a synonym measure using WordNet dictionary can discover that two concepts **Manager** and **Director** are a match. But it cannot work with concepts **Researcher** and **Reseacheur** due to a typo in **Researcheur**. In that case, a string-edit based similarity measure is more useful because the labels have many characters in common.

Intuitively, in order to deal with high terminological heterogeneity, several terminological similarity measures must be *smartly* combined in order to enhance the matching quality. Therefore, we propose a machine learning-based method to learn an intelligent combination from different terminological similarity measures. Hence, discovery of label matching can be transformed into a binary classification task [63, 64]. The main benefit of this approach is that it is flexible and self-configuring during the training process.

In order to demonstrate this idea, let's see the following example of using a decision tree learning model. For simplicity, we choose only 3 similarity measures : `Levenshtein` - a typical measure in string edit-based group; `QGrams` representing a token-based similarity measure group; and `HybLinISUB` [64]- a hybrid method combining token-based, edit-based and dictionary-based similarity measures. Those similarity measures are described in detail at [9, 47]. Of course, in YAM++, many more similarity measures belonging to those groups can be selected.

Figure 6 shows a decision tree after learning. In this example, a decision tree is a tree whose non-leaf nodes are similarity measures and whose leaf nodes are represented by a rounded rectangle shape with a value inside. This value is either 1.0 or 0.0 indicating if there is a match or not. At a non-leaf node, a similarity value of the to-be-matched entities is computed by the similarity measure stored in this node. The returned value is compared with the condition values on outgoing edges from the current node in order to decide which child node will be reached. Here, all the condition values are determined automatically by the algorithm of building the decision tree with the given training data. The classification process will start at the root node and iterates until a leaf node is reached. Edges with the condition values are indexed by numbers in pre-order traversal of the tree.

Instances	Hyb.	Lev.	QGs	CLS
Researcher|Reseacheur	0.00	0.91	0.80	?
Teacher|Lecturer	0.77	0.37	0.21	?
Manager|Director	1.00	0.13	0.10	?
teach|teaching	1.00	0.63	0.59	?

Table 1: A set of the unclassified data

Now, let us demonstrate how we use this decision tree classification in our system by several examples in Table 1. Here, we use `Hyb.`, `Lev.`, `QGs` and `CLS` as abbreviation of `HybLinISUB`, `Levenshtein`, `QGrams` and `CLASS` attributes respectively.

Let us see the feature values of the first instance, which corresponds to the pair of entities `Researcher` and `Reseacheur` from the source and target ontologies. From the root of the decision tree, the similarity score for this

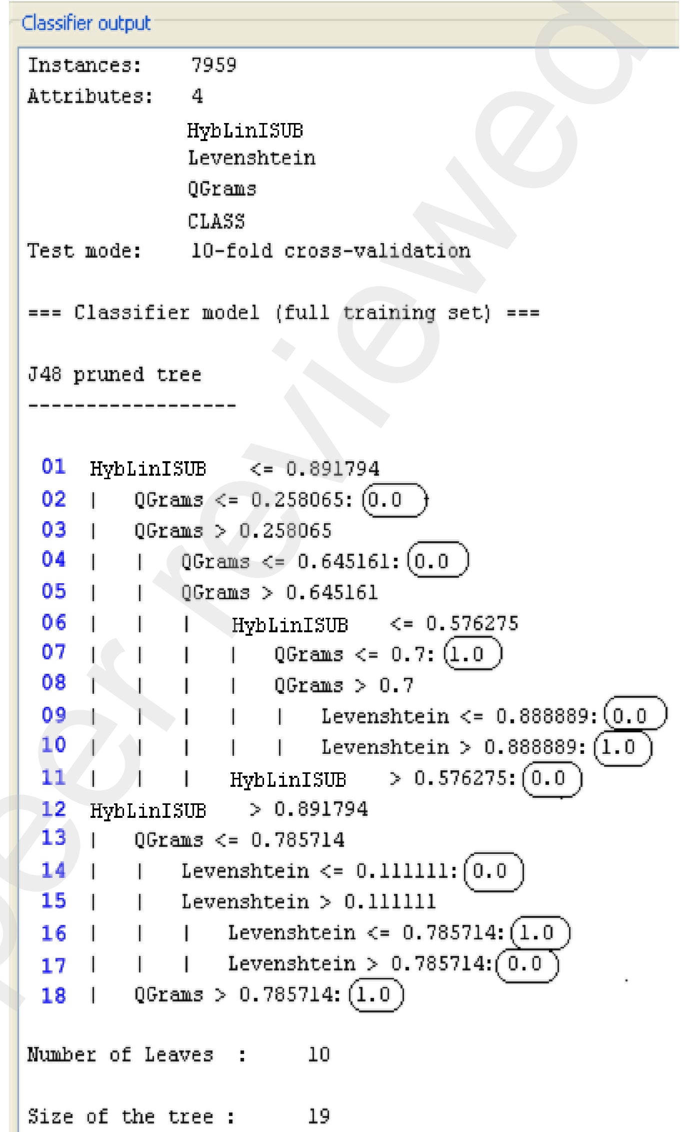


Figure 6: The trained decision tree classification

pair of entities returned by the `HybLinISUB` measure is `0.00`, which is smaller than `0.891794`. Here, `0.891794` is the condition value determined by the training process at the root node. Therefore, the decision goes through the first edge (`HybLinISUB <= 0.891794`). In the next node, QGrams returns the similarity score of 0.80, which is higher than the condition value `0.258065`. Therefore, the decision goes through the edge number 03. Similarly, this score is higher than the condition value `0.645161` in the next node, hence, the decision goes through the edge number 05. The next node is `HybLinISUB`, which returns the similarity score lower than the condition value `0.576275`. Then, the decision goes through the edge number 06 to the next node QGrams. Here, because the similarity score is higher than the condition value `0.7`, the decision goes through the edge number 08 to the `Levenshtein` node. The similarity score returned by `Levenshtein` measure is `0.91` therefore higher than the condition value `0.888889`. Then, the decision reaches the leaf with label `1.0` on the edge number `10`. It means that the entities `Researcher`

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

and Reseacheur are matched. To sum up, the edges on the path of the decision for those two entities is: 01 → 03 → 05 → 06 → 08 → 10 → leaf(1.0).

Similarly, we can apply the decision rules for the rest of unclassified data as follows. The decision path for the second instance in Table 1 is: 01 → 02 → leaf(0.0). It means the two entities Teacher and Lecturer are not a match. The decision paths for the third and fourth instances on Table 1 are the same as: 12 → 13 → 15 → 16 → leaf(1.0). These results mean that Manager matches to Director and teach matches to teaching.

Source	Target	Score
Employee	Employee	1.0
Manager	Director	1.0
Researcher	Researcheur	1.0
Subjects	Topic	1.0
hasTitle	title	1.0
teach	teaching	1.0

Table 2: Mappings discovered at the element level

The full results obtained by using this decision tree are shown in Table 2. It covers various matching results as we expected from using different terminological matchers. Broadly speaking, if we can provide an appropriate training data, the learned decision tree model, which is combining different terminological matchers, can produce a better matching results than the individual matchers [68]. However, in [66], we also showed that there exist other non-trivial types of terminological heterogeneity that cannot be resolved by terminological matchers. For example, two properties hasID and identity were not found, but we implicitly understand that they are matched because ID is supposed to be an abbreviation of identity. We will solve this issue by using structural similarity method in Section 6 and Section 7.

5.2. Information retrieval-based method

Let us see an example within a collection of ontologies describing the domain of organizing conferences [21]; people attending to the conference can be conceptualized with different names such as Conference_Participant and Attendee[7]. Those labels are highly syntactically different but they are semantically similar in the domain of conference organization. In this example, to the best of our knowledge, all terminological similarity measures will produce low similarity values for those labels. It is because these measures are mainly based only on syntactic comparison. In order to deal with this problem, we have proposed an information content-based similarity measure. In this section, we will present our approach to deal with this kind of terminological heterogeneity.

In YAM++, we have designed an information retrieval based similarity measure that uses information content of each token to compute the similarity of compound labels [75]. This idea lies on the following heuristic: if two entities are the same, the shared tokens in their labels are usually keywords and have higher information than the others have. Our proposed measure was inspired by document similarity widely used in the information retrieval field. Basically, "*stop words*", i.e., articles and prepositions, are firstly removed from documents. The remaining words are considered as informative words which convey to the content of documents. Next, a weight is assigned to each remaining word. Here, the weight value of a word represents its relative importance in the document. Finally, a computation method (e.g., cosine similarity measure) is applied to calculate a similarity score between the two given documents.

The main difference between labels comparison in the ontology matching task and the generic document comparison in information retrieval is that the former is a comparison of short strings, whereas, the latter is a comparison of long or even very long texts. Therefore, the techniques used in comparison of documents cannot be applied directly, but have to be adapted to the label comparison task. In particular, the weight assignment and the similarity computation methods should be adapted to the ontology matching context. There are many weight assignment approaches proposed in the information retrieval literature, such as, among other, term frequency (TF), inverse document frequency (IDF), the combined TFIDF, signal weighting [42]. A similar similarity method, which is based on the matching of the specification components of each entity class over synonym sets, semantic neighborhood and features like functions and attributes has been proposed in [75]. They are mainly based on statistical calculation of the frequency of occurrence of each word in a document and in a large corpus. Whereas, we consider that the weight of a word depends on the ontology that contains that word because the words used in one ontology may be different from the words used in the other. Indeed, because of their high heterogeneity, ontologies may slightly overlap or may be totally disjoint with respect to terminology. In our approach, a weight value is computed for each word appearing in a given ontology. Particularly, we apply the Shannon's information theory [78] to our weighting method. Here, a normalization of the information content of each word is considered as its weight. In this theory, the information content of an object is inversely proportional to the probability of occurrence of that object. The more times an object occurs, the less information it conveys. The information content of a word t is computed during the annotation indexing step (section 3.1).

Let s_1 and s_2 be two labels and let $Norm$ be the function, which normalizes a string by tokenizing and returns the set of composing terms of a label. Further, let $TokenSim$ be a similarity measure for two terms and let $Share(s_1, s_2) = \{t' \in Norm(s_1) \mid \exists t'' \in Norm(s_2) \wedge TokenSim(t', t'') \geq \theta\}$. By following Tversky's rule [83], we give the following definition of the similarity of two labels:

$$\text{LabelSim}(s_1, s_2) = \frac{Common_{s_1,s_2} + Common_{s_2,s_1}}{Total_{s_1} + Total_{s_2}}, \quad (5)$$

[7](in ekaw.owl and edas.owl)

Preprint not peer reviewed
This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

where, for $i, j \in \{1, 2\}$ and $i \neq j$, we have

$$Common_{s_i,s_j} = \sum_{t' \in Share(s_i,s_j)} \left(weight(t') \times \max_{t'' \in Norm(s_j)} TokenSim(t', t'') \right),$$

$$Total_{s_i} = \sum_{t \in Norm(s_i)} weight(t)$$

Let us illustrate it by an example:

$s_1 = $"$Spinal_cord_grey_matter$",
$s_2 = $"$Gray_matter_of_the_Spinal_Cord$",
$\rightarrow Norm(s_1) = \{cord\ grey\ matter\ spinal\}$,
$\rightarrow Norm(s_2) = \{cord\ gray\ matter\ spinal\}$,
$TokenSim(grey, gray) \overset{synonym}{=} 1.0$,
$\rightarrow Share(s_1, s_2) = \{cord\ grey\ matter\ spinal\} = Norm(s_1)$,
$\rightarrow Common_{s_1,s_2} = Total_{s_1}$,
$\rightarrow Share(s_2, s_1) = \{cord\ gray\ matter\ spinal\} = Norm(s_2)$,
$\rightarrow Common_{s_2,s_1} = Total_{s_2}$,
$\Rightarrow LabelSim(s_1, s_2) = 1.0$.

Our experiments showed that this measure outperforms all existing similarity measures when dealing with real-case ontology matching [66]. This method can be used alone or combined with other terminological similarity measures.

5.3. Contextual profile similarity

In addition to string-based similarity measures that are applied for labels comparison, we have implemented a contextual profile similarity measure to calculate similarity of entities through their contextual profiles. Basically, a vector space model is built from those contextual profiles, in which, each profile is represented by a vector of numerical values having the same dimension. The similarity of the two contextual profiles is determined by cosinus similarity between their corresponding vectors [64].

In YAM++, the contextual profile similarity measure has been applied at both schema level and data level. At the schema level, a context of a concept includes concept itself and a collection of concepts around such as its ancestors and descendants [64]. Then, its contextual profile is a concatenation of labels appearing in the annotation descriptions of those concepts. Our heuristic is that two concepts are similar if their contextual profiles are highly similar.

The main idea of the context-based similarity measure is described in Algorithm 1. Our approach is aimed at discovering the most similar concepts of a given one through its contextual information, such as the annotation description of the concept itself, its ancestors and descendants.

Let us present the main steps of Algorithm 1, which is dealing with context similarity measure. The input consists of two vector space models (i.e., V_s, V_t) obtained from the context indexing of the source O_s and target O_t ontologies; a given concept s of the source ontology and a parameter K being a number of the most similar concepts

Algorithm 1: Context-based Similarity Calculation

input : O_s, O_t: source and target ontologies
V_s, V_t: Context Indexing of O_s and O_t
s : a concept in the O_s
K : number of selections
output: $\{\langle t_j, v_j \rangle\}_{j=1}^{K}$: K best matches from O_t

1 $fields \leftarrow \{\mathbf{i} : individual,\ \mathbf{a} : ancestor,\ \mathbf{d} : descendant\}$;
2 **for** $f \in fields$ **do**
  // *translate contextual documents to queries*
3   $document \leftarrow$ `GetDocument`(V_s, s, f);
4   $query \leftarrow$ `BuiltQuery`$(V_t, document)$;
  // *get K best ranking scores*
5   $M_f = \{\langle t_j, r_j^f \rangle\}_{j=1}^{K} \leftarrow$ `perform`$(query, V_t, f, K)$;
6 **end**
// *get Top-K highest weighted sum*
7 $\{\langle t_j, v_j \rangle\}_{j=1}^{K} \leftarrow$ `TopK(WSum`$(M_i, w_i, M_a, w_a, M_d, w_d)$`)`;
8 **return** $\{\langle t_j, v_j \rangle\}_{j=1}^{K}$;

t_j of the target ontology O_t. The output is those K concepts, whose ranking is determined by the order of scoring value v_j returned by queries.

As we have explained in Section 3.3, each entry in the vector space model has 3 fields namely `individual`, `ancestor` and `descendant`. At line $3-4$, we convert the documents corresponding to those fields of concept s into 3 multi-terms queries with respect to the vector space model of the target ontology. More precisely, each query contains seeking information of 3 fields (ancestor, itself and descendant) Each entry in the Vector space model also contains 3 fields as above. So performing a such query means comparing the content of each field in the query to the corresponding entry in the vector space model. This comparison produces scoring values reflecting how much the query entries and those of the vector space model are similar. More details about the scoring function can be found in [48]. After executing these queries, the resulting concepts having top-K highest ranking score values will be selected. That means we obtained a map M_i (M_a, M_d) containing K concepts t_j of the target ontology, whose individual (ancestor, descendant) document is highly similar to the individual (ancestor, descendant) document of the given concept s of the source ontology (line 5).

In order to produce the final score, weighted sum ($WSum$ at line 7) is applied. Assume that concept t_j appears in all 3 maps M_i, M_a and M_d. Then the final score is calculated by $WSum$ function as follows:

$$v_j = r_j^i \times w_i + r_j^d \times w_d + r_j^d \times w_d \qquad (6)$$

If concept t_j is absent in any of the 3 maps above, the corresponding weight w_i or w_a or w_d will be set to equal 0. On the other hand, those weights are tunable and their sum is equal to 1.0.

At the data level, a contextual profile of an instance is a concatenation of its properties' textual values [60]. Let's

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

see an example in Figure 7. The two instances refer to the same object but their identification and their descriptive properties are different. However, the concatenation of the properties' values of the two instances are closely similar, thus we can assume that the two instances are matched.

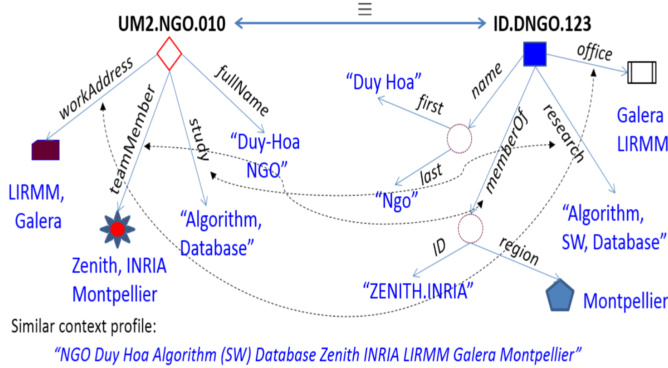


Figure 7: An example of using the extensional matcher

Once the matched instances have been detected from the two input ontologies, in YAM++, we have applied the two following heuristics: (i) Two attributes having similar values, are similar; and (ii) if two classes have highly overlapping instances, they are similar. For example, in Figure 7, we can infer that the property `workAddress` matches to the property `office`; `study` matches to `research`; and `teamMember` matches to `memberOf`.

6. Dealing with structural heterogeneity

In order to consolidate the confidence of similarity values calculated from element level, YAM++ uses structural information in a propagation process. Particularly, a derived of the Similarity Flooding (SF) algorithm [55] has been implemented in small scale ontology matching, whereas, a simple confidence propagation is used in large scale tasks.

6.1. Similarity Propagation

In small scale ontology matching, similarity propagation is used at structural level matcher. It takes the results of the element level matcher as its input and produces new similarity scores for the candidate mappings.

In this section, we present our similarity propagation (SP) method which exploits the structural information of ontologies to discover mappings. This method is inspired by the Similarity Flooding (SF) algorithm [55], which was implemented in *Rondo* schema matching tool [56].

The principle of SP is presented in Algorithm 2. First, input ontologies are transformed into directed labeled graphs, wherein each edge has the following format:

$$\langle \texttt{sourceNode}, \texttt{edgeLabel}, \texttt{targetNode} \rangle$$

Here, `sourceNode` and `targetNode` are ontological entities (i.e., concepts, properties or primitive datatypes); `edgeLabel` represents the semantic relation between entities, which is categorized into the following types: `subClass` (rdfs:subclassOf), `equivalent` (owl:equivalentClass), `subProperty` (rdfs:subPropertyOf), `inverse` (owl:inverseOf), `domain` (rdfs:domain), `range` (rdfs:range) and `onProperty` (owl:onProperty). For example, Figure 8 shows a fragment of an ontology graph around the concept of `Teacher` in the first faculty ontology depicted in Figure 2.

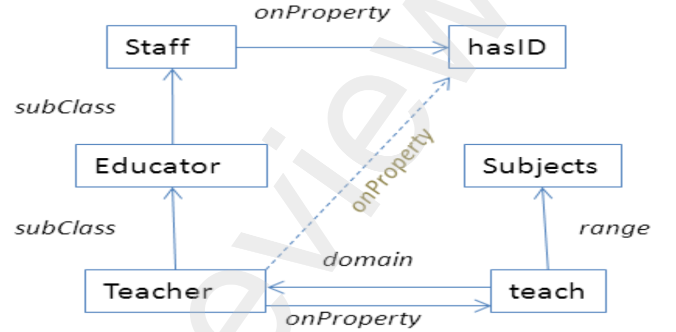


Figure 8: Fragment of an ontology graph

A pairwise connectivity graph (PCG) is then created by merging edges having the same labels. For example, if G_1 and G_2 are two graphs obtained from the transformation step, then:

$$((x, y), p, (x', y')) \in PCG \Leftrightarrow (x, p, x') \in G_1 \ \& \ (y, p, y') \in G_2$$

The intuition behind the propagation idea is that nodes of two graphs are similar when their adjacent nodes are similar. More precisely, a part of the similarity of two nodes is propagated to their neighbors which are connected by the same semantic relations. Here, the amount of similarity score to be propagated is defined by the current similarity score hold in `PCG`'s nodes and by the weight values on the edges. Therefore, at the beginning, the edges in the `PCG` are assigned with weight values by the `Weighted` function and the nodes in `PCG` are assigned with similarity values taken from the initial mappings M_0 obtained from the element-level matcher (see section 5). After initializing the values, the `PCG` becomes an induced propagation graph `IPG`. During the `Propagation` on `IPG`, only the similarity score of the nodes is changed, whereas, the edges' weights are not. At the end of each iteration in `Propagation`, all the similarity values are normalized by `Normalized` to fall in range [0,1]. When the `Propagation` meets a stop condition, a `Filter` is used to produce the mapping results.

Algorithm 2: Similarity Propagation Algorithm

input : O_s , O_t : ontologies to be matched
$M_0 = \{(e_s, e_t, \equiv, w_0)\}$: initial mappings
output: M $= \{(e_s, e_t, \equiv, w)\}$: result mappings

1. $G_s \leftarrow$ `Transform`(O_s);
2. $G_t \leftarrow$ `Transform`(O_t);
3. $PCG \leftarrow$ `Merge`(G_s, G_t);
4. $IPG \leftarrow$ `Initiate`$(PCG,$ `Weighted`$, M_0)$;
5. `Propagation`$(IPG,$ `Normalized`$)$;
6. $M \leftarrow$ `Filter`(IPG, θ_s);

In YAM++, the `Weighted` function on edge p heading from a source vertex V_s to a target vertex V_{t_i} in the

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

pairwise connectivity graph PCG is defined as follows:

$$\mathrm{w}(V_s, p, V_{t_i})) = \frac{1}{\|V_T\|} \quad (7)$$

where $V_T = \{V_{t_i}\}$ is a collection of target vertices pointed by the same edge p from the same vertex V_s. The fix-point computation used to update similarity value of vertices in the PCG in each iteration is formulated as follows:

$$\begin{aligned}\sigma^{i+1}(x,y) &= \sigma^0(x,y) + \sigma^i(x,y) \\ &+ \sum_{(a_s,p,x)\in G1,(b_s,p,y)\in G2} \sigma^i(a_s,b_s) \times w((a_s,b_s),p,(x,y)) \\ &+ \sum_{(x,q,a_t)\in G1,(y,q,b_t)\in G2} \sigma^i(a_t,b_t) \times w((x,y),p,(a_t,b_t))\end{aligned} \quad (8)$$

Here, $\sigma^i(x,y)$ is a similarity value of x and y at iteration i; $\sigma^0(x,y)$ is taken from initial mappings M_0.

After running the similarity propagation process at the structure level on the motivating example, we obtain the following mappings as shown in Table 3.

Source	Target	Score
Courses	LearningModule	0.6460724
Manager	Director	0.2716023
Researcher	Researcheur	0.2770916
Subjects	Topic	0.80278397
Staff	Employee	0.428497
Educator	AcademicStaff	0.04201378
Teacher	Lecturer	0.49652436
hasTitle	title	0.54690593
teach	teaching	0.84298825
hasID	identity	1.0

Table 3: Mappings discovered at the structure level

After running the similarity propagation process with initial mappings produced by the element level matcher (i.e., decision tree model in Section 5.1 on the motivating example, we obtain the mapping result shown in Table 3. Interestingly, it reveals new mappings as we expected to see from the semantic structure of the input ontologies. For example, it discovers $\langle$`hasID`, `identity`, $\equiv, 1.0\rangle$, which is reasonable because the both properties are used to describe attributes of many matching concepts, i.e., `Staff` vs. `Employee`, `Manager` vs. `Director`, `Teacher` vs. `Lecturer` and `Researcher` vs. `Researcheur`. During the similarity propagation process, those mappings continuously feed their impact (i.e., a portion of their similarity score) to the pair `hasID` vs. `identity`. Therefore, albeit the two properties were not detected as matched by the element level matcher, they still have the highest similarity score after the propagation process.

6.2. Confidence Propagation

In large scale ontology matching, the repetitive process of similarity propagation described in the above algorithm requires not only a huge amount of operative memory for maintaining computational results at each iteration but is also highly time consuming. The intuition behind the confidence propagation is simply that if two concepts of two ontologies representing similar domains are *matched* then their relatives are more or less similar respectively. More precisely, the similarity of the two matched concepts will contribute in some degree of confidence to the similarity of their relatives along the same path of semantic relations.

Algorithm 3: Confidence Propagation

input : $srcStrucInd$, $tarStrucInd$: source and target structure indexes
$M = \{\langle s,t,v\rangle\}$: candidate mapping table
output: $M = \{\langle s,t,v_{propagated}\rangle\}$

```
1  S ← GetSourceConcepts(M);
2  T ← GetTargetConcepts(M);
   // copy of current candidate mapping table
3  M' ← Clone(M);
4  for ⟨s_c, t_c, v_c⟩ ∈ M' do
5      for s_a ∈ GetAncestors(s, srcStrucInd) ∩ S do
6          for t_a ∈ GetAncestors(t, tarStrucInd) ∩ T
           do
7              if ∃⟨s_a, t_a, v_a⟩ ∈ M' then
8                  Update(M, ⟨s_a, t_a, v_a + v_c⟩);
9                  Update(M, ⟨s_c, t_c, v_a + v_c⟩);
10             end
11         end
12     end
13 end
14 return M
```

The confidence propagation is applied to promote the reliability of similarity of candidate mappings, which is described in Algorithm 3. The input consists of a candidate mapping table M and two structure indexes, i.e., $srcStrucInd$ and $tarStrucInd$ of the source and target ontologies respectively. Here, the structure index allows us to access quickly to all the ancestors of a given concept in a given ontology. Firstly, at the lines $1-2$, the sets S and T pick up all source and target concepts from the candidate mappings table M. They are used to reduce the number of iterations in loops at lines $5-6$. Table M' is a copy of M in order to store the current similarity values for the current candidate mappings (line 3). The iteration of the similarity propagation is shown in lines $4-12$. In particular, for each candidate mapping $\langle s_c,t_c,v_c\rangle$, if there exists another candidate mapping $\langle s_a,t_a,v_a\rangle$ in the candidate mappings table M, where s_c and t_c are descendants of s_a and t_a respectively, then the similarity values of the both candidate mappings will be updated (lines $8-9$).

7. Mapping selection

In this section we will present our method to combine the mapping results obtained at the element level with the ones resulting from the structure level. We assume that the results obtained by these levels complement each other since the element level matcher relies only on terminology (i.e., labels, annotation, etc.) while the structure level

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270
Preprint not peer reviewed

matcher relies only on semantic relations. Let us present our algorithm dealing with this combination and mapping selection. Indeed, Algorithm 4, takes as input $M_{element}$ a set of candidate mappings discovered by the element level matcher. Note that, the similarity score $c_e = 1.0$ (e.g., see Table 2). $M_{structure}$ is the set of candidate mappings gained after running the similarity propagation algorithm at the structure level. The similarity score c_s varies from `0.0` to `1.0` (e.g., see Table 3).

Algorithm 4: Producing final mappings

input : $M_{element} = \{(e_i, e_j, \equiv, 1.0)\}$
$M_{structure} = \{(e_p, e_q, \equiv, c_s),\ c_s \in (\theta_s..1]\}$
output: $M_{final} = \{(e_1, e_2, \equiv, c),\ c \in [0..1]\}$

1 $\theta \leftarrow \texttt{min}(m.c_s) \mid m \in M_{overlap}$;
2 $M \leftarrow \texttt{WeightedSum}(M_{element}, \theta, M_{structure}, (1-\theta))$;
3 threshold $\leftarrow \theta$;
4 $M_{final} \leftarrow \texttt{GreedySelection}(M, \text{threshold})$;
5 **return** M_{final}

To take the contribution of each level into account, we use a weighted sum method to combine them. Reminding the output of the element level matcher is passed as input to the structure matcher. In order to avoid manual setting, we propose a dynamic method to automatically set weights to the mappings returned by the element and structure matchers and select a threshold to filter mappings (lines 1,2 and 3 in Algorithm 4). Let us explain our method along with an example as illustrated in Figure 9 where the candidate mappings belonging to $M_{element}$ and $M_{structure}$ are indicated by labels with "`em`" and "`sm`" prefixes respectively. Additionally, we define an overlap as a mapping m that its source and target entities, i.e., $\langle e_s, e_t \rangle$ can be found in both $M_{element}$ and $M_{structure}$. Those mappings $M_{overlap}$ are labeled with "`se`" prefix.

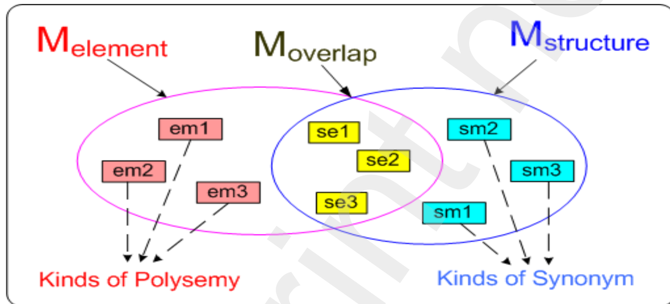


Figure 9: Resulting candidate mappings

In Figure 9, obviously, the mappings belonging to $M_{overlap}$ = {`se1`,`se2`,`se3`} have the best potential to be matched because their entities seem to have both similar name/labels and similar semantic description. Next, the mappings belonging to $M_{structure}$ = {`sm1`,`sm2`,`sm3`} are a sort of synonym because their entities seem to have different labels but have similar semantic descriptions. Whereas, each mapping belonging to $M_{element}$ = {`em1`,`em2`,`em3`} is a sort of polysemy because their entities seem to have similar name/ labels but different semantic descriptions. Intuitively, the explicit meaning of an entity (through the semantic relations with other entities) is more important than its intended meaning (through name, labels). Therefore, the order of confidence to be selected as correct mapping is: $M_{element} < M_{structure} < M_{overlap}$. In our approach, we assume that all the mappings in $M_{overlap}$ are correct.

Next, two issues arise: (i) should all the mappings in $M_{element}$ be ignored? ; and (ii) should all the mappings in $M_{structure}$ be accepted? For the first question, due to the high heterogeneity of ontologies, it is possible that entities referring to the same thing may have a different semantic description or only a small overlap between them. Therefore, we cannot definitely reject all these candidate mappings. Instead, we should assign to them a confidence value for the later selection. For the second question, we cannot accept all of them because may be their similarity scores obtained by the structure level are very small. Therefore, we need a threshold θ to filter the probably incorrect mappings. It means that if two entities have $c_s \geq \theta$ then they are probably matched.

Let us see Algorithm 4 to understand how we calculate θ value for filtering mappings from $M_{element}$ and filter threshold for mappings in $M_{structure}$. Firstly, we seek the minimum value of the structural similarity score in $M_{overlap}$ (line 1). We assume that all the mappings having a structural similarity score, which is higher than this value will be considered as correct. Therefore, we assign this value to the filter threshold θ. According to our intuition discussed above, the probability of correctness of the mappings in $M_{element}$ is smaller than the priority of mappings in $M_{structure}$, we will set the confidence to the mappings in $M_{element}$ to θ. This rule guarantees that the similarity scores of correct mappings in $M_{structure}$ are always higher than the similarity scores of correct mappings in $M_{element}$. Thus, when we perform a selection method (line 3,4), the mappings in $M_{structure}$ have higher confidence than mappings in $M_{element}$. Finally, in the $M_{overlap}$, to normalize the similarity score value, we set weights to the similarity values obtained by the element and structure levels to θ and $1-\theta$ respectively. Then we compute their similarity by the weighted sum function (line 2).

For the illustration of this idea, we continue with the motivating example. The structural matcher is based on the Similarity propagation method described in Section 6. And its input are taken from the mappings resulting from the terminological matcher.

Let us see the mapping results in Table 2 and Table 3. There, the pair of entities (`Employee`, `Employee`) has the minimum value ($c_s = 0.2716023$) found in the overlap between the results of the element level and the structure level matchers. Then, we set $\theta = 0.2716023$. Let us now illustrate our combination method along with three different candidate mappings. The first one is (`Manager`, `Director`), which is found in $M_{element}$ with a score of 1.0 and in $M_{structure}$ with a score of 0.2716023. Therefore, according to line 2 of Algorithm 4 the updated similarity score of this candidate mapping is computed as follows: $1.0*\theta + 0.2716023*(1-\theta) = 0.4694368$. The second candidate mapping is (`Course`, `LearningModule`), which is only

Preprint not peer reviewed
This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

Source	Target	Score
Courses	LearningModule	0.6460724
Manager	Director	0.4694368
Researcher	Researcheur	0.47343516
Subjects	Topic	0.8563483
Staff	Employee	0.428497
Educator	AcademicStaff	0.04201378
Employee	Employee	0.2716023
Teacher	Lecturer	0.49652436
hasTitle	title	0.6699673
teach	teaching	0.885633
hasID	identity	1.0

Table 4: Combination results of the element and structure matchers

Source	Target	Score
Courses	LearningModule	0.6460724
Manager	Director	0.4694368
Researcher	Researcheur	0.47343516
Subjects	Topic	0.8563483
Staff	Employee	0.428497
Teacher	Lecturer	0.49652436
hasTitle	title	0.6699673
teach	teaching	0.885633
hasID	identity	1.0

Table 5: Result after greedy selection with $\theta = 0.2716023$

found in $M_{structure}$. Therefore, its similarity score is still be the same. The third candidate mapping is (`Employee`, `Employee`), which is only found in $M_{element}$ with a score of 1.0; its update similarity score is: $1.0 * \theta = 0.2716023$. Table 4 shows the results obtained by combining the element level candidate mappings with the ones resulting from structure level matcher. Then, a Greedy Filtering with threshold $\theta = 0.2716023$ is applied on these mappings to obtain the final result as shown in Table 5. It is remarkable to see that our method discards the candidate mapping (`Employee`, `Employee`), which got a score of 1.0 in element level.

8. Dealing with alignment incoherence

In this section, we present our approach to eliminate the unsatisfiability from the discovered mappings. In term of small ontology matching, YAM++ has integrated ALCOMO [53] tool, which uses Pellet[8] (or Hermit[9]) reasoning to completely diagnose and remove unsatisfiable mapping candidates. However, those reasoners cannot work with large scale ontologies with a small memory size (e.g., below 4GB RAM) or run quite slow while performing conflict detection, thus, we propose a new method called **fast semantic filtering** in YAM++.
Our approach is aimed to effectively detect and remove conflicting pairs of mappings by exploiting the semantic information of entities in the input ontologies. In particular, three patterns including disjointedness (i.e., subclass over disjoint and disjoint over subclass) conflicts [53], criss-cross conflict [36] have been reused. For example, assume that candidate mapping $\langle Employee, Employee \rangle$ was not filtered after the mapping selection process, then it will cause a disjointness conflict with $\langle Manager, Director \rangle$. Indeed, in the target ontology, the concept `Director` is a subclass of the `Employee`, consequently, in the source ontology, the concept `Manager` becomes a subclass of the concept `Employee`, but the concept `Manager` is a subclass of the concept `Staff`, which is disjoint to the concept `Employee`, thus causing an inconsistency.

On the other hand, for large scale ontology matching, a new relative disjoint conflict [60] has been proposed. The definition of a relative disjoint conflict pattern is that if two mappings $m1 = \langle e_{s_1}, e_{t_1} \rangle$ and $m2 = \langle e_{s_2}, e_{t_2} \rangle$ have $e_{s_1} \equiv e_{s_2}$ in O_s and $SemSim(e_{t_1}, e_{t_2}) \leq \theta$ in O_t (or vice versa) then they are detected as in relative conflict. Here, $SemSim$ is a similarity measure that computes the semantic similarity value of two entities in the same ontology [62]. This pattern is inspired from learning disjointness axioms [54, 77] in case where the input ontologies do not explicitly declare any disjointness axiom.

In order to quickly detect conflict patterns from the candidate mappings, we have proposed an efficient method to index the structure of an ontology. Indeed, in our approach, each concept is assigned with a *topological order* and is encoded by an efficient compressed bitmap [85, 10] containing all the information of its ancestors, descendants and disjointness [62]. Thanks to the bitmap encoding table, the structural relations of classes can be easily and quickly accessed.

Algorithm 5: FAST SEMANTIC FILTERING METHOD

input : O_s, O_t : input ontologies,
A : initial alignment
output: A_c : coherent alignment

```
1 DescendingSortByConfidenceValues(A);
2 A_c ← ∅
3 while ||A|| > 0 do
4     m ← extractFirstMapping(A);
5     if isNotConflict(m, A_c, O_s, O_t) then
6         A_c ← A_c ∪ {m}
7     end
8 end
```

After all conflict mappings have been detected, in OAEI 2012, YAM++ used a greedy algorithm to break those conflicts. Indeed, it iteratively found and removed a candidate mapping that conflicts with other candidates and has the smallest confidence value. The main steps of the process to eliminate the inconsistent candidate mappings is shown in Algorithm 5. Firstly, it sorts all the mappings in the initial alignment A by confidence value in descending order. Then, for each iteration, it picks up the mapping with the highest confidence value from the initial alignment A. At line 5, if the examined mapping m does not cause any conflict to the already extracted alignment A_c, it will be saved in A_c. In the example above, mapping

[8]http://clarkparsia.com/pellet
[9]http://www.hermit-reasoner.com/

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

Preprint not peer reviewed

⟨*Employee*, *Employee*⟩ is chosen to be removed. Afterward, there is no more conflict in the discovered mappings.

In OAEI 2013, YAM++ improved its performance by applying a modification of Clarksons algorithm in finding a minimum weighted vertex cover from the collection of conflicts [58]. In this approach, it not only relies on the similarity score of the candidate mappings (i.e., weighted nodes in the conflict graph), but also how strong (i.e., degree of nodes in the graph) each mapping conflicts to the others. The detail of the algorithm can be found in [62].

9. Experiments over OAEI datasets

In this section we present the evaluation results of YAM++ in various OAEI tracks and in different years.

9.1. Filtering Evaluation

Firstly, we present the important role of the filtering in reducing the computational space. Let's demonstrate its effectiveness through an example where the size of the used fragments of ontologies are shown in Table 6. If we do not run any filtering method, we need to compute similarity values for `89676425` candidate mappings in `Library` track, `5270462036` candidate mappings in `FMA-NCI` track, `9673308896` candidate mappings for `FMA-SNOMED` track and `8171287936` candidate mappings for `SNOMED-NCI` track. Note that each concept in these ontologies may have some labels. The average number of labels per concept are shown in Table 6. Thus, for example, in task `FMA-NCI`, we need to perform the computation of more than `31` billions similarity values. Running similarity computation with a simplest edit-based string metric (e.g., Levenstein) in a PC i5core 3.20GHz, we observe that it can compute in average 100000 pairs of labels in `01` second. Thus, to compute similarity for `31 billions` pairs of labels, we need at least `87` continuous hours. Moreover, if we use dictionary-based (e.g., Wordnet) similarity metric to compute similarity value of two labels, the running time is much more longer. The huge number of computations requires a huge main memory for saving temporary similarity values as well as a very long time to complete this computation. That is the one reason that not many ontology matching tools can pass the large biomedical ontology matching track.

	STW	TheSoz	NCI	FMA	SNOMED
Size	6575	13639	66724	78989	122464
Avr.Labels	5	3	3	2	1

Table 6: Size of ontologies and number of labels per concept

In order to reduce the computation space, we have applied a filtering step before computing similarity values between concepts. Note that, in the filtering method described in Section 4, one label can produce several sublables by removing its non-stop words. Here, a stop-word is an article or a preposition like *a*, *an*, *in*, *on*, etc. In our approach, we have applied the following heuristic in the filtering step: "If two labels of two concepts differ by less than two non-stop words then the two concepts are considered similar". Therefore, from an original label of a concept we produce its sub-labels by removing only `01` non-stop word.

Task	Pr.	Re.	Fm.
Library	0.00954	0.97691	0.0189
FMA-NCI	0.00712	0.96588	0.01413
FMA-SNOMED	0.01269	0.79409	0.02498
SNOMED-NCI	0.01031	0.81717	0.02036

Table 7: Precision, Recall and F-measure after filtering

Table 7 shows the evaluation results after performing filtering on the `Library`, `FMA-NCI`, `FMA-SNOMED`, `SNOMED-NCI` tasks. In this step, we focus more on **recall** than on **precision**. It is because the **recall** value indicates the number of correct mappings that the matching tool can discover. The **precision** value indicates the accuracy of the discovered mappings, which will be improved afterward by step-by-step. The experimental results are quite impressive. For the `Library` and `FMA-NCI` tasks, we obtained very high recall, i.e., `0.97691` and `0.96588` respectively, whereas, for the `FMA-SNOEMD` and `SNOMED-NCI` the recall values are lower but still high, i.e., `0.79409` and `0.81717` respectively.

Task	#TP	#FP	#FN	#All pairs
Library	3088	320507	73	89676425
FMA-NCI	2831	395003	100	5270462036
FMA-SNOMED	7100	552454	1841	9673308896
SNOMED-NCI	15098	1449216	3378	8171287936

Table 8: Number of True Positive, False Positive and False Negative after filtering

Next, Table 8 shows the number of candidate mappings obtained from the filtering step. For the Library task, the total number of candidate mappings is `323595` including `3088` correct mappings and `320507` incorrect mappings. In comparison with all possible mappings (shown in the last column - `89676425`), the filtering reduces $\frac{8967425}{323959} \approx$ 28 times of computation. Similarly, the filtering reduces nearly `13248`, `17288`, `5580` times of computation for `FMA-NCI`, `FMA-SNOMED`, `SNOMED-NCI` tasks respectively. Obviously, these reductions significantly decrease the computational time to calculate similarity values for candidate mappings. For example, in case of FMA-NCI task, if we use a simple edit-based similarity metric like Leveinstein, we only need `03` seconds.

Library	FMA-NCI	FMA-SNOMED	SNOMED-NCI
64347 ms	260597 ms	320139 ms	359085 ms

Table 9: Preparation time including indexing and filtering steps

On the other hand, Table 9 shows the total time starting from indexing ontologies to producing candidate mappings in the filtering step. The Library task needs only `65` seconds, whereas, the SNOMED-NCI requires nearly `01` minute to complete the filtering task. Indeed, the preparation time is much smaller than the total time needed to compute similarity values for all possible mappings (for example `87` hours in case of FMA-NCI).

By considering all the benefits including high recall values, significantly reduced computational space and small preparation time, we can firmly conclude that our pro-

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

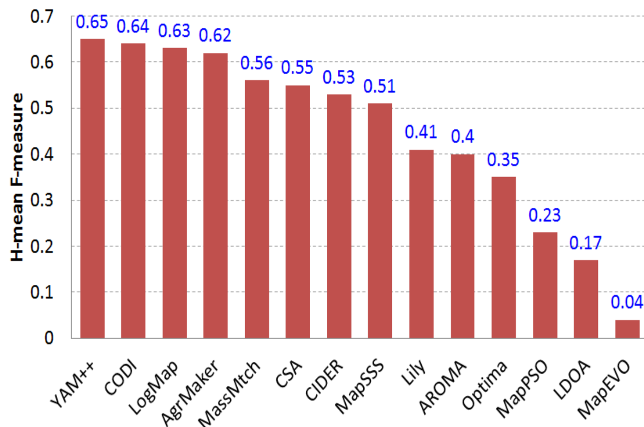


Figure 10: Results of YAM++ version using machine learning in Conference track at OAEI 2011

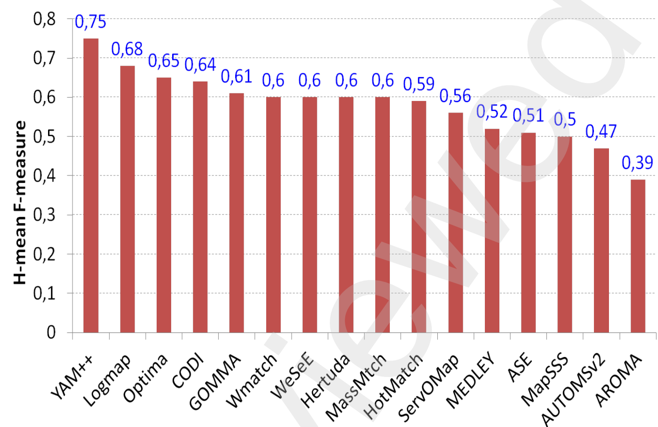


Figure 11: Results of YAM++ version without using machine learning in Conference track at OAEI 2012

posed filtering method is compulsory and extremely important in dealing with large scale ontology matching.

9.2. Conference track: testing the heterogeneity

Conference track contains 16 real ontologies describing the same domain about conference organization. This track is open+blind due to only 21 referent alignments corresponding to the complete alignment between 7 ontologies (Cmt, ConfTool, Edas, Ekaw, Iasted, Sigkdd, Sofsem). Those ontologies are highly heterogeneous in terms of both terminology and structure. They are very good examples for testing techniques dealing with terminological and structural heterogeneity. Figure 10 shows the results of YAM++ on Conference track in OAEI 2011, where YAM++ used machine learning method to combine terminological similarity measures.

In OAEI 2011, YAM++ used machine learning approach to combine different string-based similarity measures to deal terminological heterogeneity. We have used the dataset taken from I3CON[10] as the training data to build a decision tree. YAM++ got in average 0.78 for *Precision*, 0.56 for *Recall* and 0.65 for *Fmeasure*.

Despite the fact that YAM++ obtained a high *Precision* and outperforms the other participants in terms of the best *Fmeasure*[11] its *Recall* value is only above average. On the other hand, the quality of the classification model strongly depends on the training data, which is not widely available. Therefore, in OAEI 2012, we decided to experiment a new approach by using our new IR-based method described in Section 5.2 to deal with terminological heterogeneity. More discussion and comparison between the two proposed approaches, can be found in [68]. Figure 11 shows the results of YAM++ version 2012 on the same track without using machine learning. It is noticeable that in this track YAM++ achieved significantly higher *Fmeasure* 0.75 value than the previous one and the other participants. Moreover, this version also achieved the best *Precision* 0.78 as well as the *Recall* 0.65. We made the choice to show OAEI 2011 result in Conference track because it illustrates the use of machine learning as a combination method. This method is no more used in the further versions of YAM++ in OAEI. Moreover, YAM++ got the same F-measure value in OAEI 2013 for this track than in OAEI 2012 due to the absence of new updates in terminological matcher.

[10] http://www.atl.external.lmco.com/projects/ontology/i3con.html
[11] http://oaei.ontologymatching.org/2011/results/conference/

9.3. Multifarm track

More interestingly, the effectiveness of the terminological matcher and propagation method can be viewed in the Multifarm track. Indeed, the original Multifarm data set is composed of a set of 7 ontologies of the Conference domain (Cmt, Conference, ConfOf, Edas, Ekaw, Iasted, Sigkdd), translated into 8 languages (+English) – Chinese (cn), Czech (cz), Dutch (nl), French (fr), German (de), Portuguese (pt), Russian (ru), Spanish (es) – and the corresponding cross-lingual alignments between them.

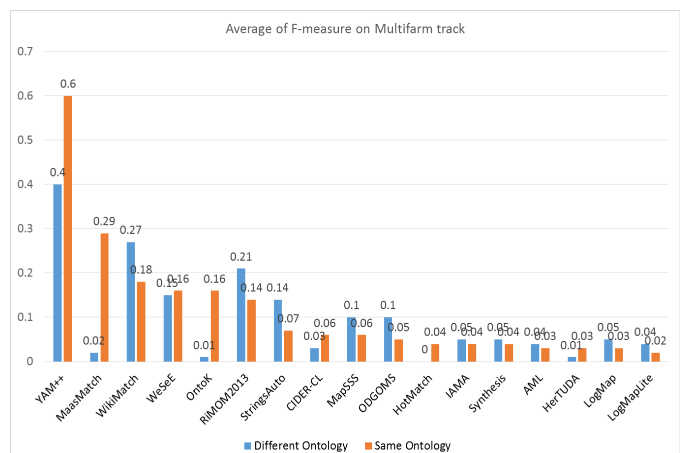


Figure 12: Comparison of YAM++ with other participants in Multifarm track OAEI2013

Within the Multifarm dataset, we can distinguish two types of matching tasks: (i) those test cases where two different ontologies have been translated in different languages (cmt–confOf, for instance); and (ii) those test cases where the same ontology has been translated in different languages (cmt–cmt, for instance). On the both type

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

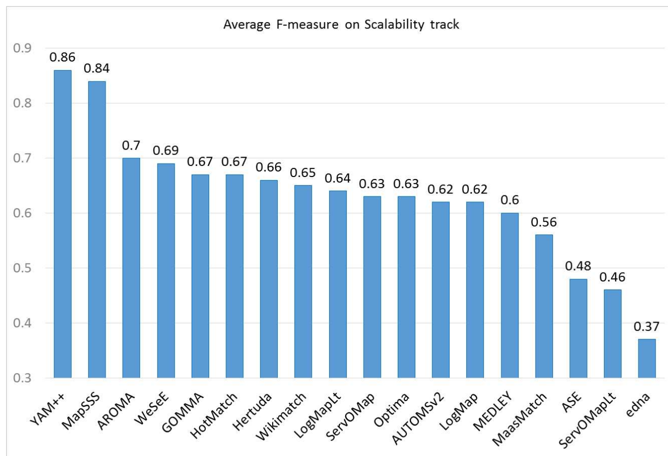


Figure 13: Comparison of YAM++ with other participants in Scalability track OAEI 2012

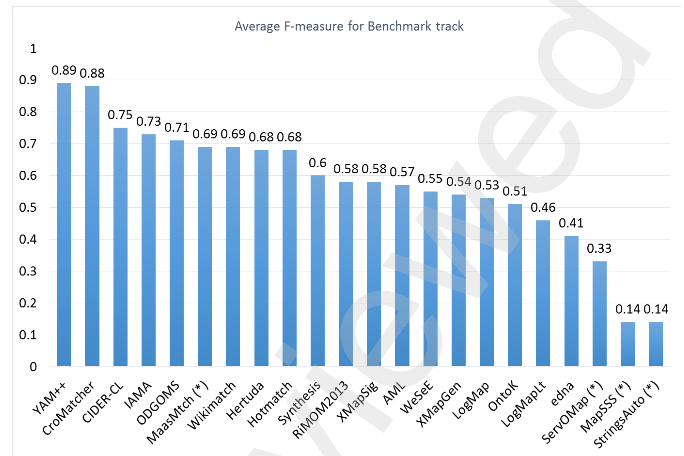


Figure 14: Comparison of YAM++ with other participants in Benchmark track OAEI 2013

of matching task, since the first competition,i.e., OAEI 2011.5 to OAEI 2013 YAM++ achieved the best results. Figure 12 shows how much YAM++ outperformed the other competitors in this track.

9.4. Scalability track

The test of scalability was not provided in OAEI 2013. Therefore, we report here the result of the one of OAEI 2012. Benchmark or Scalability track 2012 includes 5 datasets with increasing size: biblio (97 entities), hidden benchmark 2 (247 entities), hidden benchmark 3 (354 entities), hidden benchmark 4 (472 entities), finance (633 entities). This track was aimed at evaluating the strengths and weaknesses of ontology matchers through a set of systematically generated tests.

In fact, for each dataset, the test ontologies are generated from a reference ontology by altering labels, annotation and structure of its entities. Most of the altered labels are random sequence of characters, which is not convenient for human reading. However, the data of those ontologies are not changed much, thus our extensional matcher can be very useful. Indeed, Figure 13 shows the overall evaluation results of YAM++ and the other participants. YAM++ got the best F-measure results in 4/5 datasets, particularly, **benchmark2** (0.89), **benchmark3** track (0.85), **benchmark4** (0.83), **finance** (0.90), and was the second in **biblio** (0.83) In average, YAM++ achieved the first position with 0.86 *F-measure*. This result confirms the stability and the scalability of our tool over years thanks to the effectiveness of our matching techniques namely the effectiveness of the filtering methods and the proposed similarity measures for dealing with terminological and structural heterogeneity.

9.5. Benchmark track

The Benchmark track in OAEI 2013, includes only one test set suite which is generated from the usual bibliography ontology. The test case of Benchmark was not available to participants. In fact, the test ontologies are generated from a reference ontology by altering the labels, annotation and structure of its entities. Figure 14 shows the overall evaluation results of YAM++ and of the other participants. In average, YAM++ got the first position with 0.89 *Fmeasure*. Moreover, as mentioned by the OAEI organizers, YAM++ was the fastest in this track. it still performs in less than 12 minutes for 94 matching operations. This result confirms the robustness and the improvement of our tool over years thanks to the effectiveness of our matching techniques namely the IR-based measure, the *extensionnal* and the *propagation* methods. The gap between YAM++ and its competing systems, namely: ServOMap, AML and LogMap is quite impressive in terms of matching quality. Indeed, AML got only 0.57 *Fmeasure*. It seems that the reason is that AML does not handle instances and the matching techniques used at element level are not so effective. LogMap got 0.53 and ServOMap 0.33 in *Fmeasure*.

9.6. Library track

The library track is a real-word task to match the STW and the TheSoz social science thesauri in SKOS. The goal of this track is to find whether the matchers can handle these lightweight ontologies including a huge amount of concepts and additional descriptions. In OAEI 2013, YAM++ got the second position behind ODGOMS in terms of F-measure (0.745 against 0.758). This year, ODGOMS and YAM++ outperformed all baselines; further these matchers outperformed all baselines even MatcherPref[12]. While ServOMap and AML are behind them. However, our tool outperforms ODGOMS in terms of running time with a great margin (around 12 min against more than 7 h). ODGOMS combines different matching techniques (e.g., LCS, TFIDF based, Stoilos, etc.). This system got good results in small sized ontology matching scenario but this combination seems to not be scalable in large scale scenario. Indeed, it did not pass even the small large bio medical task since its execution time exceeded the maximum limit of 18 hours. Furthermore, YAM++ has improved its performance in this track both in terms

[12] http://web.informatik.uni-mannheim.de/oaei-library/results/2013

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

of Recall (from 0.758 in OAEI 2012 to 0.808 in OAEI 2013) and in Precision, which also increased from 0.680 to 0.692. This behavior can be explained by the annotation features associated with concepts in the ontologies. Indeed, each concept in the Library ontologies has several labels. STW contains 6575 concepts with in average 5 labels per concept. While TheSoz has 13639 concepts with in average 3 labels per concept. The rich annotation will provide a high contextual similarity score for the similar concepts. Therefore, the good results of YAM++ are due to the effectiveness of our contextual and IR-based similarity measures.

9.7. Anatomy track

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. Figure 15 shows the evaluation result of YAM++ in this track.

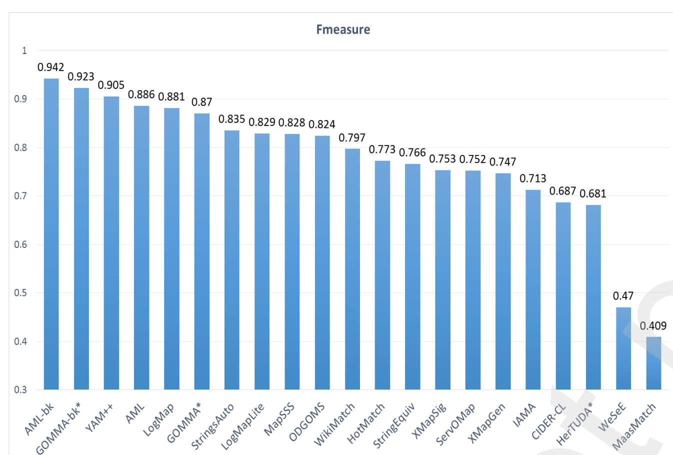


Figure 15: Results of YAM++ and of the other participants to Anatomy track at OAEI 2013

As we can see in Figure 15, two top ranked systems AML-bk and GOMMA-bk outperform YAM++ in this track. The main reason is the following: both AML-bk and GOMMA-bk use mapping composition techniques and the reuse of mappings between UMLS, Uberon and FMA. Notice that UMLS and Uberon contains a lot of mappings between Mouse anatomy and Human anatomy, which cannot be solely discovered by using common string similarity measures. Nevertheless, YAM++ got the third position (with 0.905 *Fmeasure*) in this track thanks to the effectiveness of our filtering method and the IR-based similarity measure, which have been applied to the labels and sub-labels of the entities of the Anatomy ontologies. Indeed, this measure was designed to efficiently handle compound labels and it is very effective thanks to its particular weight assignment method. While the competing systems are behind YAM++. Indeed, AML [19] got 0.886, LogMap 0.881 and ServOMap got 0.752. we notice that these systems make use of Background Knowledge. For example, AML employs three sources of background knowledge: Uberon, UMLS and WordNet.

9.8. Large biomedical track

This track consists of finding alignments between the Foundational Model of Anatomy (FMA), SNOMED CT, and the National Cancer Institute Thesaurus (NCI). There are 6 tasks with different size of input ontologies, i.e., small fragment, large fragment and the whole ontologies.

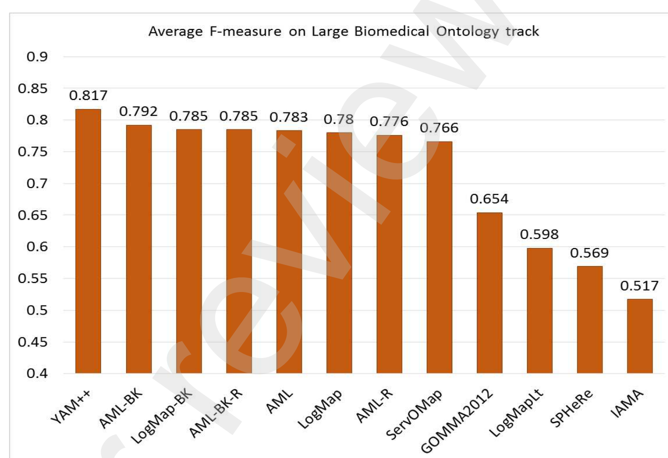


Figure 16: Accuracy performance in large biomedical track

Figures 16 and 17 shows the evaluation results and the time performance of YAM++ in large biomedical track in OAEI 2013. YAM++ achieved the first position with 0.817 *Fmeasure*. Our system outperformed its competing systems, which were designed to large scale biomedical matching, namely: ServOMap, AML and LogMap in terms of matching quality. YAM++ used indexing and filtering techniques in many places (in filtering candidate mappings, semantic coherence checking, etc.) as the competing proposals did. The success of YAM++ is due to the combination of several contributions like our effective IR-based measure and our dynamic method for mapping selection, which generates dynamically a threshold according to the matching scenario. While, for example, in ServOMap the threshold is not dynamically generated. And regarding the run time, YAM++ outperforms ServOMap: $2,066$ for YAM++ against $15,300$s for ServOMap.

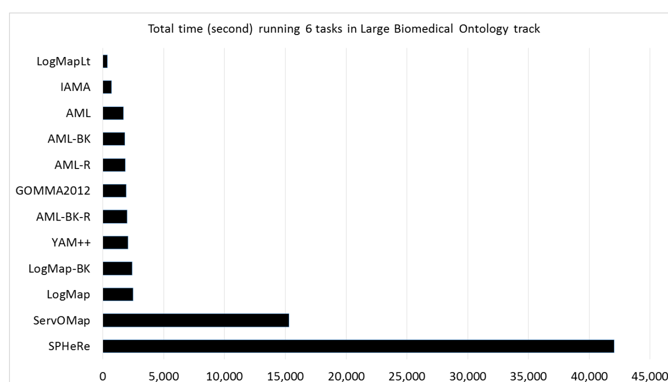


Figure 17: Run time performance in large biomedical track

Furthermore, YAM++ was among the three systems: AML with (R)epair configuration and LogMap that have shown mapping repair facilities. Finally, we can notice a great improvement of YAM++ namely regarding the

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

time performance. Indeed, our systems spent 67,817 for executing the large biomedical track in OAEI 2012 while it took only 2,066 in OAEI 2013. The improvement was done without lost of matching quality in terms of F-measure since YAM++ got 0.782 in OAEI 2012 against 0.817 in OAEI 2013. This is mostly the benefit gained by indexing and filtering techniques.

10. Related work

The aim of this section is to provide a brief survey of related work that are close to ours. We will try to compare the benefit of our solutions to other competing works. Therefore, we will focus only on the following issues: (i) workflow strategies, (ii) efficiency and scalability and (iii) Matcher combination, (iv) Mapping Selection and (v) External Background Knowledge

10.1. Workflow strategies

Many basic matching techniques have been proposed so far, but none of them can fulfill the user requirement in order to obtain a full picture of alignments between ontologies. Commonly, each of them only provides partial mappings according to the specific exploited feature. Therefore, strategies to combine several basic techniques (or basic/individual matcher) are needed. According to [24], there are three basic types of workflow strategies such as *sequential*, *parallel* and *interactive* composition. Most of the currently matching system have implemented at least one of those strategies to improve the efficiency and the matching quality. In this section, we present those basic strategies and see how they were used in the matching tools.

Sequential workflow. It is the most natural way of composing basic matchers. The idea is that the output of one basic matcher is passed as input to the next matcher. This type of combination can be found in many systems. For example, most of the structural or semantic matcher require predefined mappings. To do that, an element level matcher first produces initial mappings and then passes them to the structural matcher. This type of strategy can be found in CUPID [49], Falcon AO [33], Rondo [56], where a terminological measure is the first matcher and produces initial mappings to the second matcher (structural matcher); or in CODI [35], PRIOR++ [52] the first matcher aims to find the all possible candidate mappings, then in the next phase, a semantic matcher refines them by eliminating the inconsistent mappings. In ServOMap [14], at element level, a weighted sum of the three following measures is applied : ISub, Q-Gram and Levenshtein and then the obtained mappings are passed to *Contextual Similarity* module. This module is machine learning-based and makes use of descendant, ancestor and sibling relationships to discover new mappings. AgreementMakerLight (AML) [19] is a light version of AgreementMaker ontology matching system [20], which is dedicated as to deal with large ontology matching problem namely in live sciences domain. Its main underlying matching techniques are based on element level matching with a Lexical Matcher and the use of three background knowledge: Uberon, UMLS and WordNet. In YAM++, as we show in Figure 1, the sequential workflow starts from element level, then structure level and finally semantic verification. In 2011 version of YAM++, we applied a machine learning method to select and to combine the most appropriate similarity measure at element level. In the current version, YAM++ used our IR-based similarity measure at element level and the propagation method at structure level.

Parallel workflow. In this strategy, the individual matchers are executed independently, typically on the whole cross product of source and target entities. Similarity matrices obtained by individual matchers are combined by some aggregation operators to produce a final result. The most common operators are weighted sum or weighted average. In that way, the systems may be implemented with manual assignment weights (COMA [2], ASCO [45]) or with adaptive weights (the first phase in PRIOR++ [52], AgreementMaker [20]) or fuzzy assignment weights (OWA [37]). Besides, aggregation operators can be seen as a decision functions which decide whether two entities are matched or not. In that way, the individual matchers are combined by machine learning models, in which the object is a pair of to-be-matched entities and attributes are similarity values obtained from individual matchers over the given entities. The systems of this type of combination can be found in [72, 57]. Similarly, the belief theory Dempster-Shafer framework DDSim [59] and evolutionary methods MapPSO-MapEVO [5] are also used as aggregation operators. In YAM++, parallel workflow can be found in the machine learning part, which is used to combine different terminological similarity measures.

Iterative workflow. The idea of this strategy is that the matching process runs repeatedly several times until it meets the stop condition. The matching process can be performed by one basic matcher or by a combination of several basic matchers. This type of strategy can be used in a part of the system or in the whole system. A typical example of using iterative method in a part of the system can be found in the similarity propagation method. The principle of the algorithm is that the similarity between two nodes must depend on the similarity between their adjacent nodes. Therefore, at each step of the running algorithm, the similarity value of each pair of entities is recomputed according to the current values of itself and its neighbors. We can find this strategy in the second phase of emphRondo schema matching tool [56], OLA [22], Falcon-AO [33], Lily [84], RiMOM [46], AgreementMaker [20]. Similarly, the iterative strategy is also used in constraint-based method. In that way, for each step, the constraint-based method re-calculates the confidence values for every candidate mapping or removes the inconsistent mappings. This process will be stopped when the optimization condition is reached. The iterative constraint-based methods can be found in the second phase of GLUE [15], CODI [35], LogMap [38]. In *Rondo* schema matching tool [56] [55], the interative workflow can be found at the structure level, i.e., similarity propagation part.

Typical examples of using iterative workflow in the

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

whole system can be found in ASMOV [36], QOM [16]. In those systems, the matching process consists of several basic matching techniques. For example, in the first phase of ASMOV, they use parallel composition of terminological, structural and extensional measures, then pass the discovered mappings to the second phase to run the semantic verification process. Those two phases are repeated several times until there is no change found in the discovered mappings.

10.2. Efficiency and scalability techniques

High matching quality is the prime importance that needs to be considered in the development of any ontology matching tool. Besides, efficiency in terms of performance time is also very important, especially, when the input ontologies are very large whereas the main memory size is limited. To deal with large-scale ontology matching, several techniques have been proposed and categorized as follows.

Filtering methods. The main idea behind these techniques is to reduce the search space by heuristically eliminating less promising matching entity pairs. For example, in Eff2Match [8], the heuristic to select candidate mappings for each entity in the source ontology is done by performing the top-K entities algorithm in the target ontology according to their context (Virtual Document) similarity. In QOM [16], more sophisticated strategies, which are based on different extracted features such as label, hierarchy, neighbors, etc., are used in each iteration to select the promising mappings. An inverted lexical index is used in LogMap 2 [40] to store concept labels and URIs of each input ontology. This allows to filter an initial set of mappings. The followed strategy for computing candidate mappings at this stage by Log Map 2 is to maximize recall. More recently, in ServOMap [14], the input ontologies (modeled as virtual documents) are indexed and then the search component of ServO is used to perform an exact search respectively for I_1 using O_2 as search component and for I_2 using O_1. Then, an intersection of the two resulting sets is done to keep all pairs found from the two way search. From the intersected results, only the Best-k results (k chosen empirically) are selected and then passed as input to the lexical matcher.

In YAM++, we have implemented various techniques based on search engine and some heuristics of filtering (by label, context profile, etc.) to reduce the computational space.

Partitioning methods. In this section, we discuss the main features of a partition-based method dealing with large scale ontology matching. The algorithm used in this method is similar to the divide and conquer algorithm. Here, the task of matching large ontologies is broken down into smaller problems (i.e., matching on sub-ontologies). In this method, two main sequential phases work as follows. Firstly, the input ontologies are partitioned into sub-ontologies. It can be done by applying a clustering algorithm overall entities of ontologies. For example, in [1, 34], two Agglomerative Hierarchical Clustering algorithms namely ROCK and SCAN have been implemented. At the end of this phase, entities within the same cluster are strongly related to each other, whereas, entities of different clusters are weakly related. Thus, we can assume that sub-ontologies built on those clusters are representing different and independent sub-domains of knowledge. Next, in order to reduce the run-time, a block (i.e., sub-ontology) filter selects candidate blocks, which will be compared to find mappings between their entities. It can be done by using some similarity measure at block level. The intuition is that if two blocks describe the same or close topic, they may share a high number of concepts and vocabulary used to represent entities on that topic. This intuition leads to two methods for computing similarity between blocks. The underlying idea of the first method is that the more anchors can be found between two blocks, the more similar the two blocks are. Here, an anchor is defined as a pair of entities, which have a high similarity value. For example, this method is implemented in [34]. The underlying idea of the second method lies in the similarity of block documents that contain the name, label of all entities of blocks. For example, this method is implemented in [1]. The computation of the similarity values between two block documents is similar to the computation of the similarity between two virtual document in Section 5.3. The advantages of this method is that it can be used as an upper layer of existing matching methods which can produce a high matching quality but it is time consuming (e.g., graph matching method). In that case, an existing matching tool can be used to discover mappings between selected pairs of blocks of the input ontologies. If the size of two blocks is small enough, the matching process can run fast and does not require extra memory. Nevertheless, this method suffers from several weaknesses. Firstly, the marginal entities within a block may loss semantic information. It is because when the method breaks down the input ontology, some relations will be cut off. Therefore, it may cause inaccuracy in the computation of similarity. Next, the semantic coherence in each block and its size strongly depend on the cut-off criteria of the clustering algorithm. There is no guarantee about the maximum size of blocks. It may produce unbalanced blocks, where some of them have a very big size and the others have small size. In that case, a big size block should be broken down again. However, the more breaking down operations are, the more information will be lost. Finally, the complexity of the clustering algorithm is high. For example, the complexity of the ROCK algorithm is $O(n^2 log(n))$; the complexity of SCAN algorithm in a worst case is $O(n^2)$, where n is the size of the ontology to be partitioned.

A sub-class of this category is known as **anchor-based partitioning** methods. These methods are a modification of the methods above, where the partitioning of to-be-matched ontologies is done according to a set of anchors. This method is also known as a dynamic selection of candidates. It does not break input ontologies into smaller partitions like previous method. Instead, it iteratively updates the set of candidate mappings, i.e., generate new candidates by exploiting structural information of entities and remove probable mismatched ones by judging their

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

similarity values. This method can be found in QOM [16], Anchor-PROMPT [70] systems, and more recently in the Anchor-Flood [31]. Basically, the main steps of this the dynamic segmentation method work as follows. Firstly, at least one of initial anchors between the input ontologies is discovered by running a fast similarity measure (e.g., an equality string-based measure). In each iteration, an aligned pair is selected to be explored. Note that the first aligned pair comes from the initial anchor. For each concept in the selected pair, the Anchor-Flood algorithm will update the ontology segment, which this concept belongs to, by adding its neighboring concepts such as super concepts, siblings and sub-concepts of certain depth. The idea is that the neighbors of similar concepts might also be similar. Then, the alignment process (i.e., similarity computation and mapping selection) produces aligned pairs from the collected neighbors. The new aligned pairs are then used as anchors for the next iteration. The process is repeated until "*either all the collected concepts are explored, or no new aligned pair is found*". It outputs a set of aligned pairs within two segments across the input ontologies. The main advantage of this method is time efficiency and memory efficiency. It does not compute similarity values for all pairs of entities in the input ontologies, but only within two "*segments*" build around the selected anchor. The complexity analysis of this method shows that it performs $O(Nlog(N))$ number of comparison in average, where N is the size of input ontologies. Because this method run iteratively, so in each iteration, the memory usage is small. However, this method may suffer from some drawbacks when the size of two input ontologies is very large and the position of aligned pairs are highly distributed. For example, the Foundational Model of Anatomy ontology (FMA.owl - 78,989 classes) and the National Cancer Institute Thesaurus ontology (NCI.owl - 66,724 classes) have in common only 2898 aligned pairs [39], which is much smaller than the size of the both ontologies. In that case, in the two segments built around a selected aligned pair, the number of aligned pairs is usually much smaller than the number of unaligned pairs. Therefore, in the alignment process, the structural similarity value computed for each pair of concepts between two segments is small; consequently, it may not discover new aligned pairs for the next iteration. This problem causes the loss of many candidates. That is maybe a reason why Anchor-Flood obtained a not high Recall (0.682) in the OAEI 2008 Anatomy track.

10.3. Matcher combination

In [24], many combination methods have been proposed to aggregate similarity values of different individual matchers. For example, the *Max/Min* methods returns the maximal/minimal similarity value of individual matchers. The *Weighted* method computes a weighted sum of similarity values of individual matchers. The *Average* method is one special case of the Weighted method and returns where weights assigned to all individual matchers are equal. The *SIGMOID* method combines multiple results using a sigmoid function, which is essentially a smoothed threshold function. Generally, *Weighted* and *SIGMOID* methods need to manually set aggregation weights based on experience for different individual matchers or tentatively factor in the sigmoid function. This way of setting parameters is not able to adapt to different matching tasks because it might work well in a specific matching scenario but not in the others. Moreover, manual setting is not flexible and nor scalable namely when the selected matchers change or when their number increases. To the best of our knowledge, there are two automatic weighted sum methods that have been implemented and proved their success. The first method is *Harmonic Adaptive Weighted Sum*, which has been introduced in the PRIOR+ system. According to the comparison analysis in [52], this method outperforms all the methods mentioned above. The second method is called *Local Confidence Weighted Sum*, which is the core method for combining individual matchers in the AgreementMaker system [11]. Finally, the selection of the final candidates from the set of candidates is performed by a greedy selection strategy to select the best candidates based on their scores. Most of the ontology matching systems use a manual setting threshold. For example, SErvOMap used a manual threshold assignment [14]. Our first contribution presented in this paper is a machine learning-based method, which is used to combine terminological similarity measures (Section 5.1). The benefit of using machine learning methods is that they can be flexible and self-configuring during the training process. The second contribution is a *Dynamic Weighted Sum* method, which is used to combine terminological and structural matchers (Section 7). For a given matching scenario, this method evaluates the degree of reliability of these matchers, and assigns appropriate weight values to them. In addition, it automatically determines a threshold value to select the final mappings result.

10.4. Mapping selection

Mapping selection is used at the end of the matching process in order to produce a final alignment result. The aim of this phase is to eliminate the suspicious incorrect mappings and the inconsistent ones. Basically, three criteria, namely, similarity values, matching cardinality and semantic consistency, are widely used in mapping selection. They lead to the corresponding three types of filtering methods such as Threshold Filter, Cardinality Filter and Semantic Filter. Many ontology matching systems have integrated a semantic verification component to remove inconsistency from the alignment but only few can really work with large scale ontology matching. In S-Match [26], this task is translated into a satisfaction problem where all ontological axioms are transformed into Conjunctive Normal Form (CNF). It then makes use of the SAT solver (Sat4J[13] library) to find the optimal solution for a consistent alignment. Similarly, in CODI [35], the authors formulate ontological axioms into a set of first order formula with weights. Then, a Markov logic framework is applied to solve the optimal consistency solution. The strategy

[13] http://www.sat4j.org/

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

followed by GOMMA [27] and by ASMOV [36] consists to discover conflicts among mappings iteratively and then to reject suspicious mappings having low similarity score at the end of each running step. More recently, justification-based mapping repair techniques have been proposed and implemented in some systems such as Alcomo [53] and LogMap [39]. They consider the integration of the two ontologies and the set of mappings between them as a single ontology and they assume that both of the ontologies are themselves logically consistent. Alcomo computes all justifications for all unsatisfiable classes while LogMap only computes one for each unsatisfiable class in each iteration until the whole set of mappings is cleaned. In AML, The repair process consists in removing incoherence-causing mappings from the input alignment. For scalability sake, their algorithm computes core fragments from the input ontologies; then detects all disjointness-based incoherences. Finally, it removes the set of mappings mappings according to a set of predetermined set of criteria (e.g., filtering the conflicting sets that have a lowest-confidence mapping) [76]. LogMap-BK [39] removes the inconsistency by using existing specialized background knowledge such as based on the UMLS lexicon [14]. In YAM++ version 2012, we have implemented a greedy-based algorithm similar to the Alcomo efficient algorithm, to perform inconsistency removing step. Whereas, in the current version, we have implemented a new inconsistency removing algorithm based on Clarkson algorithm for the weighted minimum vertex cover problem. More precisely, we proposed a semantic-based similarity measure to calculate a similarity value of two concepts in the same ontology. It allows to discover disjointness relationships in case where they are not explicitly declared in the ontology. Finally, a method that can detect and remove inconsistent mappings was implemented in YAM++. As shown in [62], our approach is especially efficient in large scale ontology matching.

10.5. External Background Knowledge

The main matching techniques that are used in GOMMA-BK [27] to deal with large ontology matching are blocking, parallel matching and mapping composition. Parallel ontology matching is performed on multiple computing nodes and CPU cores. The blocking technique allows to reduce search space by restricting matching to the overlapping ontology parts. Furthermore, GOMMA-BK allows the improvement of matching quality by using domain knowledge and mapping composition via domain-specific hub ontologies namely UMLS [6], Uberon [17] and FMA[15]. Notice that UMLS and Uberon contain an important set of mappings between Mouse anatomy and Human anatomy, which are difficult to be discovered by using common string similarity measures. Furthermore, GOMMA-BK makes use of multi-language translation services for improving the synonyms discovery. AgreementMakerLight (AML) [19] is a new and light version of AgreementMaker ontology matching system [18], which is dedicated to dealing with large ontology matching problem. The most important structure for matching is the Lexicon, a table of class names and synonyms in an ontology, which uses a ranking system to weight them and score their matches. Its main underlying matching techniques are based on element-level matching and the use of three external resources as background knowledge: Uberon, UMLS and WordNet. The main difference of YAM++ is that it is not using specialized background knowledge.

[14] http://www.nlm.nih.gov/research/umls/

[15] http://sig.biostr.washington.edu/projects/fm/AboutFM.html

11. Conclusion

In this paper, we present our approach called YAM++ to enhance ontology matching, by using techniques coming from different fields such as Machine Learning, Information Retrieval and Graph Matching. The novelty lies at individual matchers as well as at their combination level. For example, to deal with the terminological heterogeneity, we have designed a new similarity measure based on Information Retrieval techniques and a similarity propagation method to deal with structural heterogeneity. Then, a dynamic weighted sum is applied to combine and select the final mappings resulting from the element and structure matchers. Obviously, the current version of YAM++ passed all scalability and large scale ontology matching tests and obtained high matching results over OAEI datasets. The success of YAM++ is due to the combination of several contributions like our effective IR-based measure and our dynamic method for mapping selection, which generates dynamically a threshold according to the matching scenario.That is also the benefit of extensively using index structure in many places and candidate filtering techniques.

Finally, YAM++ is publicly available [16]. In the very near future, we will make available a multitask platform including an online version of YAM++. Furthermore, a GUI for the offline version for YAM++ is under development.

We are currently working on the use of Background in YAM++.

References

[1] Alsayed Algergawy, Sabine Massmann, and Erhard Rahm. A clustering-based approach for large-scale ontology matching. In *ADBIS*, 2011.

[2] David Aumueller, Hong Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *SIGMOD*, pages 906–908, 2005.

[3] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[4] Fernando Bobillo, Miguel Delgado, and Juan Gómez-Romero. Representation of context-dependant knowledge in ontologies: A model and an application. *Expert Syst. Appl.*, pages 1899–1908, 2008.

[5] Jürgen Bock, Carsten Dänschel, and Matthias Stumpp. MapPSO and MapEVO results for OAEI 2011. In *OM*, 2011.

[6] Olivier Bodenreider. The unified medical language system (UMLS): integrating biomedical terminology. *Nucleic Acids Research*, 32(Database-Issue):267–270, 2004.

[16] http://www.lirmm.fr/yam-plus-plus

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

[7] Abdelghani Chibani, Karim Djouani, and Yacine Amirat. Semantic middleware for context services composition in ubiquitous computing. In *MOBILWARE*, page 9, 2008.
[8] Watson Wei Khong Chua and Jung-Jae Kim. Eff2Match results for OAEI 2010. In *OM*, 2010.
[9] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.
[10] Alessandro Colantonio and Roberto Di Pietro. Concise: Compressed 'n' composable integer set. *Information Processing Letters*, pages 644–650, 2010.
[11] Isabel F. Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. Efficient selection of mappings and automatic quality-driven combination of matching methods. In *OM*, 2009.
[12] Isabel F. Cruz and Huiyong Xiao. Using a layered approach for interoperability on the semantic web. In *WISE*, pages 221–231, 2003.
[13] Ricardo de Almeida Falbo, Fabiano Borges Ruy, and Rodrigo Dal Moro. Using ontologies to add semantics to a software engineering environment. In *SEKE*, pages 151–156, 2005.
[14] Gayo Diallo. An effective method of large scale ontology matching. *J. Biomedical Semantics*, 5:44, 2014.
[15] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Y. Halevy. Ontology matching: A machine learning approach. In *Handbook on Ontologies*, pages 385–404. Springer-Verlag, 2004.
[16] Marc Ehrig and Steffen Staab. QOM – quick ontology mapping. In *ISWC*, pages 683–697. Springer, 2004.
[17] C. J. Mungall et al. Uberon, an integrative multi-species anatomy ontology. In *Genome Biology*, volume 13, 2012.
[18] Cruz Isabel F. et al. Using agreementmaker to align ontologies for oaei 2010. In *OM*, 2010.
[19] Daniel Faria et al. AgreementMakerLight results for OAEI 2013. In *OM*, 2013.
[20] Isabel F. Cruz et al. Using AgreementMaker to align ontologies for OAEI 2010. In *OM*, 2010.
[21] Svab et al. OntoFarm: Towards an experimental collection of parallel ontologies. In *Poster Session at ISWC*, 2005.
[22] Jérôme Euzenat, David Loup, Mohamed Touzani, and Petko Valtchev. Ontology alignment with OLA. In *Proceedings of the 3rd EON Workshop, 3rd International Semantic Web Conference*, pages 59–68, 2004.
[23] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn dos Santos. Ontology alignment evaluation initiative: Six years of experience. *J. Data Semantics*, pages 158–192, 2011.
[24] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.
[25] Avigdor Gal and Pavel Shvaiko. Advances in ontology matching. In *Advances in Web Semantics I*, pages 176–198. Springer-Verlag, Berlin, Heidelberg, 2009.
[26] Fausto Giunchiglia and Pavel Shvaiko et al. S-Match: an algorithm and an implementation of semantic matching. In *ESWS*, pages 61–75, 2004.
[27] Anika Gross, Michael Hartung, Toralf Kirsten, and Erhard Rahm. GOMMA results for OAEI 2012. In *OM*, 2012.
[28] Ramanathan V. Guha, Rob McCool, and Eric Miller. Semantic search. In *WWW*, pages 700–709, 2003.
[29] Alon Y. Halevy. Why your data won't mix: Semantic heterogeneity. *ACM Queue*, pages 50–58, 2005.
[30] Fayçal Hamdi, Brigitte Safar, Nobal B. Niraula, and Chantal Reynaud. TaxoMap alignment and refinement modules: results for OAEI 2010. In *OM*, 2010.
[31] Md. Seddiqui Hanif and Masaki Aono. Anchor-Flood: Results for OAEI 2009. In *OM*, 2009.
[32] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.
[33] Wei Hu, Jianfeng Chen, Gong Cheng, and Yuzhong Qu. ObjectCoref & Falcon-AO: results for OAEI 2010. In *OM*, 2010.
[34] Wei Hu, Yuzhong Qu, and Gong Cheng. Matching large ontologies: A divide-and-conquer approach. *Data Knowl. Eng.*, pages 140–160, 2008.
[35] Jakob Huber, Timo Sztyler, Jan Nößner, and Christian Meilicke. CODI: Combinatorial optimization for data integration: results for OAEI 2011. In *OM*, 2011.
[36] Yves R. Jean-Mary and Mansur R. Kabuka. ASMOV: Results for OAEI 2008. In *OM*, 2008.
[37] Qiu Ji, Peter Haase, and Guilin Qi. Combination of similarity measures in ontology matching using the OWA operator. In *Recent Developments in the Ordered Weighted Averaging Operators, : Theory and Practice*, Studies in Fuzziness and Soft Computing, pages 281–295. Springer-Verlag, 2011.
[38] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. LogMap: Logic-based and scalable ontology matching. In *ISWC*, pages 273–288, 2011.
[39] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Yujiao Zhou. LogMap 2.0: Towards logic-based, scalable and interactive ontology matching. In *International Workshop on Semantic Web Applications and Tools for the Life Sciences*, pages 45–46, 2012.
[40] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau Yujiao Zhou, and Ian Horrocks. Large scale interactive ontology matching: Algorithms and implementation. In *ECAI*, 2012.
[41] M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In *IJCAI'01*, 2001.
[42] Gerald Kowalski. *Information Retrieval Systems: Theory and Implementation.* Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1997.
[43] Andreas Langegger, Wolfram Wöß, and Martin Blöchl. A semantic web middleware for virtual data integration on the web. In *ESWC*, pages 493–507, 2008.
[44] Holger Lausen, Ying Ding, Michael Stollberg, Dieter Fensel, Rubén Lara Hernandez, and Sung-Kook Han. Semantic web portals: state-of-the-art survey. *J. Knowledge Management*, 9(5):40–49, 2005.
[45] Bach Thanh Le, Rose Dieng-Kuntz, and Fabien Gandon. On ontology matching problems. In *ICEIS (4)*, pages 236–243, 2004.
[46] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. RiMOM: A dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng.*, pages 1218–1232, 2009.
[47] Feiyu Lin and Kurt Sandkuhl. A survey of exploiting WordNet in ontology matching. In *IFIP AI*, pages 341–350, 2008.
[48] McCandless M, Hatcher E, and Gospodnetic O. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0.* Manning Publications Co, Greenwich, CT, USA, 2010.
[49] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with Cupid. In *VLDB*, pages 49–58, 2001.
[50] Alexander Maedche, Steffen Staab, Nenad Stojanovic, Rudi Studer, and York Sure. SEmantic portAL: The SEAL approach. In *Spinning the Semantic Web*, pages 317–359, 2003.
[51] Christoph Mangold. A survey and classification of semantic search approaches. *Int. J. Metadata Semant. Ontologies*, pages 23–34, 2007.
[52] Ming Mao, Yefei Peng, and Michael Spring. An adaptive ontology mapping approach with neural network based constraint satisfaction. *J. Web Sem.*, 8(1):14–25, 2010.
[53] Christian Meilicke. Alignment incoherence in ontology matching. In *Thesis*, 2011.
[54] Christian Meilicke, Johanna Völker, and Heiner Stuckenschmidt. Learning disjointness for debugging mappings between lightweight ontologies. In *Knowledge Engineering EKAW*, pages 93–108, 2008.
[55] 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.
[56] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. RONDO: A programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, 2003*, pages 193–204, 2003.
[57] Prasenjit Mitra, Natasha F. Noy, and Anuj Rattan Jaiswal. OMEN: a probabilistic ontology mapping tool. In *ISWC*, 2005.
[58] Rajeev Motwani. Lecture notes on approximation algorithms: Volume i. Technical report, 1993.
[59] Miklos Nagy, Maria Vargas-Vera, and Piotr Stolarski. DSSim results for OAEI 2009. In *OM*, 2009.
[60] DuyHoa Ngo and Zohra Bellahsene. YAM++ results for OAEI 2012. In *OM*, 2012.

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270

[61] DuyHoa Ngo and Zohra Bellahsene. Yam++ results for oaei 2013. In *OM*, 2013.
[62] DuyHoa Ngo and Zohra Bellahsene. Efficient semantic verification of ontology alignment. In *ACM IEEE WI*, pages 141–148, 2015.
[63] DuyHoa Ngo, Zohra Bellahsene, and Remi Coletta. A flexible system for ontology matching. In *CAiSE Forum (Selected Papers)*, pages 79–94, 2011.
[64] DuyHoa Ngo, Zohra Bellahsene, and Remi Coletta. A generic approach for combining linguistic and context profile metrics in ontology matching. In *OTM Conferences (2)*, pages 800–807, 2011.
[65] DuyHoa Ngo, Zohra Bellahsene, and Remi Coletta. YAM++ results for OAEI 2011. In *OM*, 2011.
[66] DuyHoa Ngo, Zohra Bellahsene, and Konstantin Todorov. Extended tversky similarity for resolving terminological heterogeneities across ontologies. In *ODBASE*, pages 711–718, 2013.
[67] DuyHoa Ngo, Zohra Bellahsene, and Konstantin Todorov. Extended tversky similarity for resolving terminological heterogeneities across ontologies. In *OTM Conferences*, pages 711–718, 2013.
[68] DuyHoa Ngo, Zohra Bellahsene, and Konstantin Todorov. Opening the black box of ontology matching. In *ESWC*, pages 16–30, 2013.
[69] DuyHoa Ngo, Zohra Bellasene, and Remi Coletta. A generic approach for combining linguistic and context profile metrics in ontology matching. In *ODBASE Conference*, 2011.
[70] Natalya F. Noy and Mark A. Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *IJCAI*, pages 63–70, 2001.
[71] Natalya Fridman Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.
[72] Rong Pan, Zhongli Ding, Yang Yu, and Yun Peng. A bayesian network approach to ontology mapping. In *ISWC*, pages 563–577, 2005.
[73] Shvaiko Pavel and Jerome Euzenat. Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 99, 2013.
[74] Velma L. Payne and Douglas P. Metzler. Hospital care watch (hcw): An ontology and rule-based intelligent patient management assistant. In *CBMS*, pages 479–484, 2005.
[75] M. Andrea Rodriguez and Max J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 15:442–456, 2003.
[76] Emanuel Santos, Daniel Faria, Catia Pesquita, and Francisco M. Couto. Ontology alignment repair through modularization and confidence-based heuristics. *PLoS ONE*, 10:e0144807, 12 2015.
[77] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In *The Semantic Web: Research and Applications*, pages 27–44. Springer, 2005.
[78] C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, pages 50–64, 1951.
[79] Pavel Shvaiko and Jérôme Euzenat. Ten challenges for ontology matching. In *OTM Conferences (2)*, pages 1164–1182, 2008.
[80] Paul R. Smart and Paula C. Engelbrecht. An analysis of the origin of ontology mismatches on the semantic web. In *EKAW*, pages 120–135, 2008.
[81] Vassilis Spiliopoulos, George A. Vouros, and Vangelis Karkaletsis. On the discovery of subsumption relations for the alignment of ontologies. *Web Semant.*, pages 69–88, 2010.
[82] Steffen Staab, Jürgen Angele, Stefan Decker, Michael Erdmann, Andreas Hotho, Alexander Maedche, Hans-Peter Schnurr, Rudi Studer, and York Sure. Semantic community web portals. *Computer Networks*, 33(1-6):473–491, 2000.
[83] Amos Tversky. Features of similarity. *Psychological Review*, 84:327–352, 1977.
[84] Peng Wang. Lily results on SEALS platform for OAEI 2011. In *OM*, 2011.
[85] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Optimizing bitmap indices with efficient compression. *ACM Trans. Database Syst.*, pages 1–38, 2006.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=3199270