# Enhancing Ontology Matching by Using Machine Learning, Graph Matching and Information Retrieval Techniques

Duy Hoa Ngo

UNIVERSITÉ MONTPELLIER II
— SCIENCES ET TECHNIQUES DU LANGUEDOC —

THÈSE

pour obtenir le grade de
Docteur de l'Université Montpellier II

| | |
|---|---|
| DISCIPLINE | : INFORMATIQUE |
| *Spécialité Doctorale* | : *Informatique* |
| *Ecole Doctorale* | : *Information, Structure, Systèmes* |

présentée et soutenue publiquement par

DuyHoa Ngo
le 12 Décembre 2012

# Enhancing Ontology Matching by Using Machine Learning, Graph Matching and Information Retrieval Techniques

## Abstract

In recent years, ontologies have attracted a lot of attention in the Computer Science community, especially in the Semantic Web field. They serve as explicit conceptual knowledge models and provide the semantic vocabularies that make domain knowledge available for exchange and interpretation among information systems. However, due to the decentralized nature of the semantic web, ontologies are highly heterogeneous. This heterogeneity mainly causes the problem of variation in meaning or ambiguity in entity interpretation and, consequently, it prevents domain knowledge from sharing. Therefore, ontology matching, which discovers correspondences between semantically related entities of ontologies, becomes a crucial task in semantic web applications.

Several challenges to the field of ontology matching have been outlined in recent research. Among them, selection of the appropriate similarity measures as well as configuration tuning of their combination are known as fundamental issues that the community should deal with. In addition, verifying the semantic coherent of the discovered alignment is also known as a crucial task. Furthermore, the difficulty of the problem grows with the size of the ontologies.

To deal with these challenges, in this thesis, we propose a novel matching approach which combines different techniques coming from the fields of machine learning, graph matching and information retrieval in order to enhance the ontology matching quality. Indeed, we make use of information retrieval techniques to design new effective similarity measures for comparing labels and context profiles of entities at element level. We also apply a graph matching method named similarity propagation at structure level that effectively discovers mappings by exploring structural information of entities in the input ontologies. In terms of combination similarity measures at element level, we transform the ontology matching task into a classification task in machine learning. Besides, we propose a dynamic weighted sum method to automatically combine the matching results obtained from the element and structure level matchers. In order to remove inconsistent mappings, we design a new fast semantic filtering method. Finally, to deal with large scale ontology matching task, we propose two candidate selection methods to reduce computational space.

All these contributions have been implemented in a prototype named YAM++. To evaluate our approach, we adopt various tracks namely Benchmark, Conference, Multifarm, Anatomy, Library and Large Biomedical Ontologies from the OAEI campaign. The experimental results show that the proposed matching methods work effectively. Moreover, in comparison to other participants in OAEI campaigns, YAM++ showed to be highly competitive and gained a high ranking position.

**Keywords**

Ontology matching, Information retrieval, Machine learning, Similarity propagation, Semantic verification

TITRE en français : **Amélioration de l'alignement d'ontologies par les techniques d'apprentissage automatique, d'appariement de graphes et de recherche d'information**

## Resumé

Ces dernières années, les ontologies ont suscité de nombreux travaux dans le domaine du web sémantique. Elles sont utilisées pour fournir le vocabulaire sémantique permettant de rendre la connaissance du domaine disponible pour l'échange et l'interprétation au travers des systèmes d'information. Toutefois, en raison de la nature décentralisée du web sémantique, les ontologies sont très hétérogènes. Cette hétérogénéité provoque le problème de la variation de sens ou ambiguïté dans l'interprétation des entités et, par conséquent, elle empêche le partage des connaissances du domaine. L'alignement d'ontologies, qui a pour but la découverte des correspondances sémantiques entre des ontologies, devient une tâche cruciale pour résoudre ce problème d'hétérogénéité dans les applications du web sémantique. Les principaux défis dans le domaine de l'alignement d'ontologies ont été décrits dans des études récentes. Parmi eux, la sélection de mesures de similarité appropriées ainsi que le réglage de la configuration de leur combinaison sont connus pour être des problèmes fondamentaux que la communauté doit traiter. En outre, la vérification de la cohérence sémantique des correspondances est connue pour être une tâche importante. Par ailleurs, la difficulté du problème augmente avec la taille des ontologies.

Pour faire face à ces défis, nous proposons dans cette thèse une nouvelle approche, qui combine différentes techniques issues des domaines de l'apprentissage automatique, d'appariement de graphes et de recherche d'information en vue d'améliorer la qualité de l'alignement d'ontologies. En effet, nous utilisons des techniques de recherche d'information pour concevoir de nouvelles mesures de similarité efficaces afin de comparer les étiquettes et les profils d'entités de contexte au niveau des entités. Nous appliquons également une méthode d'appariement de graphes appelée propagation de similarité au niveau de la structure qui découvre effectivement des correspondances en exploitant des informations structurelles des entités. Pour combiner les mesures de similarité au niveau des entités, nous transformons la tâche de l'alignement d'ontologie en une tâche de classification de l'apprentissage automatique. Par ailleurs, nous proposons une méthode dynamique de la somme pondérée pour combiner automatiquement les correspondances obtenues au niveau des entités et celles obtenues au niveau de la structure. Afin d'écarter les correspondances incohérentes, nous avons conçu une nouvelle méthode de filtrage sémantique. Enfin, pour traiter le problème de l'alignement d'ontologies à large échelle, nous proposons deux méthodes de sélection des candidats pour réduire l'espace de calcul.

Toutes ces contributions ont été mises en IJuvre dans un prototype nommé YAM++. Pour évaluer notre approche, nous avons utilisé des données du banc d'essai de la compétition OAEI : Benchmark, Conference, Multifarm, Anatomy, Library and Large Biomedical Ontologies. Les résultats expérimentaux montrent que les méthodes proposées sont très efficaces. De plus, en comparaison avec les autres participants à la compétition OAEI, YAM++ a montré sa compétitivité et a acquis une position de haut rang.

**MOT-CLES**
alignement d'ontologies, extraction/recherche d'information, apprentissage automatique, propagation de similarité, vérification sémantique.

# Acknowledgements

I extend my sincere gratitude and appreciation to many people who made this Ph.D. thesis possible. First of all, I want to express my gratitude to my family and friends for their encouragement during all these years. Their support to pursue this goal has been enormous.

I wish to thank my supervisor Zohra Bellahsene for being my mentor in research. Her consistent confidence in me to publish research had propelled me to where I stand today.

I would like to thank all my thesis jury members. Special thanks to Yamine Ait-Ameur and Jérôme Euzenat for their valuable comments and remarks as reviewers of the dissertation. I am indebted to Marie-Laure Mugnier, Manuel Ruiz and Rémi Coletta for taking out time from their hectic schedules, to act as examiners for my work.

I owe a lot of thanks to Konstantin Todorov for his valuable feedback and his help in my English writing. I am also very grateful to Imen Mami for her help in many administrative procedures in France. Special thanks to my friends and colleagues at LIRMM and GALERA for their informal discussions and support.

Finally, this Ph.D. would not have been possible without support from INRIA, University Montpellier II and Centre National de la Recherche Scientifique (CNRS) of France. My stay in France for pursuing the doctorate degree has been financially supported by the ANR DataRing ANR-08-VERSO-007-04.

Montpellier, October 2012

Duy-Hoa NGO

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Ontology matching is an active research field, which is a key solution for solving the semantic heterogeneity problem. The performance of an ontology matching system including matching quality and runtime efficiency plays an important role in the success of many semantic applications. In this chapter, we introduce the main topics that we are concerned with in the field of ontology matching throughout the thesis. We start by presenting the role of ontology and the requisite of ontology matching in the computer science field (Section 1.1). Next, we discuss the challenging issues in the ontology matching field (Section 1.2). The objectives of our research and our contributions are then stated in Section 1.3 and Section 1.4 respectively. Finally, Section 1.5 presents the structure of this thesis

## 1.1 Ontology, Ontology Matching and Semantic Web

In recent years, ontologies have attracted a lot of attention in Computer Science, especially in the Semantic Web field. In its original meaning in philosophy, ontology is concerned with the fundamental question of "*what kinds of things are there?*" and leads to studying general categories for all things that exist in a domain of interest [22]. In general, it is visualized and thought of as a *semantic network* that displays interrelated conceptual nodes. For example, in a technical vehicular domain, to describe a concept `Car`, people relate it to a sub-category of the concept `Vehicle` which is a sub-category of the concept `Engine`.

Transferring from philosophy to computer science, *an ontology* is a computational artifact that encodes knowledge of this domain in a machine-processable form to make it available to information systems. It is a *formal explicit specification of a shared conceptualization of a domain of interest* [34]. The term *formal* in this definition implies that an ontology is based on the grounds of formal logic to represent knowledge. The term *explicit* implies that ontologies have to state knowledge explicitly to make it accessible for machines. This ensures that the specification of domain knowledge in an ontology is machine-processable and is being interpreted in a well-defined way. The term *specification* means an ontology represents knowledge about a particular domain of interest. The narrower the scope of the domain is, the more detail about its concepts can be captured in the ontology. Finally, the term *shared conceptualization* implies that an ontology reflects an agreement on a domain conceptualization among people in a community.

Owning the important characteristics discussed above, ontologies can serve as explicit conceptual knowledge models and provide the semantic vocabularies 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 [35, 61], semantic portal [89, 60, 55], semantic information integration [16, 73, 5], intelligent advisory systems [77, 6], semantic middleware [54], semantic software engineering [18].

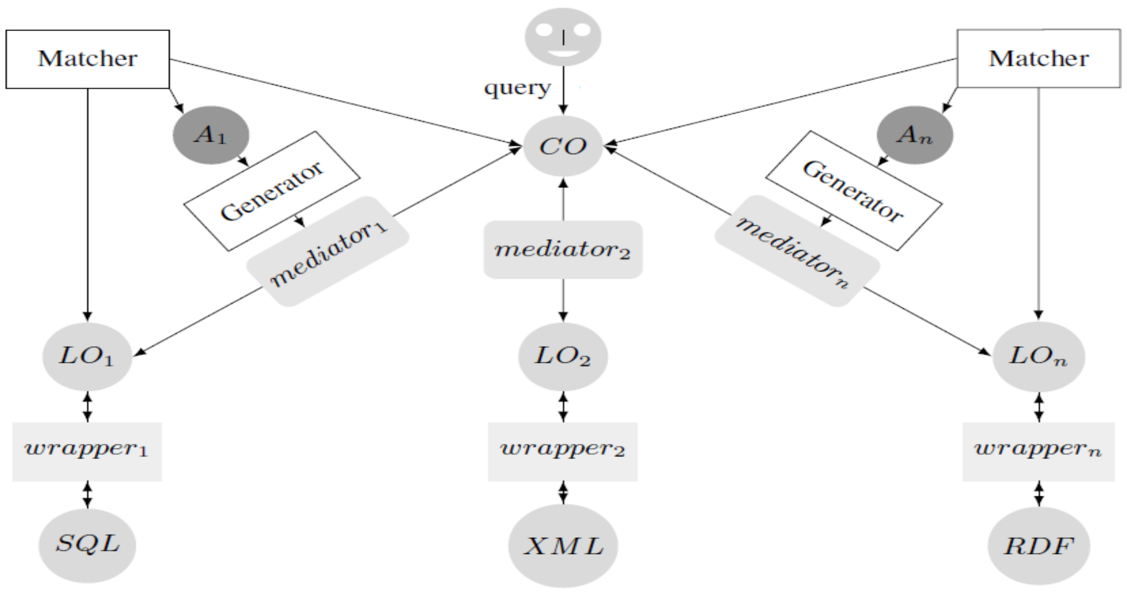


Figure 1.1: A scenario of semantic information integration [28]

As we discussed above, the strength of an ontology is to support sharing knowledge between information systems. Now, let us illustrate a general scenario of semantic information integration in Fig. 1.1 in order to understand the role, the position of an ontology and the requisite of ontology matching in a semantic application. This scenario is taken from the Ontology Matching book by Euzenat and Shvaiko [28]. Assume that data are stored in different local information sources in different formats like SQL, XML, RDF, etc. An information integration task is to gather those data in order to provide the users a unique query interface via a global (or common) ontology ($CO$) to all the local information sources. To do that, each local information source is wrapped to local ontology ($LO_i$), which then are matched against the global ontology. The alignment between them ($A_i$) helps generate a $mediator_i$ which in turn transforms queries against the common ontology into a query to the information source and translates the answers in the other way.

This is only one example of semantic application that involves ontology and ontology matching. Further applications of ontologies and ontology matching are found in ontology engineering, information integration, including schema integration, catalogue integration, data warehouses and data integration, peer-to-peer information sharing, web service composition, autonomous communication systems, including agents and mobile devices communication, and navigation and query answering on the web.

Obviously, a key for the success of these applications is related to the matching operation between ontologies. However, it is one of the most difficult issues because of the high heterogeneity of ontologies [96, 50, 36]. Due to the de-centralized nature of the semantic web, an explosion of the number of ontologies has been produced. Many of them may describe similar domains, but they are very different because they have been designed independently by different ontology developers following diverse modeling principles and patterns. For example, within a collection of ontologies describing the domain of organizing conferences [93], people attending to the conference can be conceptualized with different names such as `Participant` (in confOf.owl), `Conference_Participant` (in ekaw.owl), `Attendee` (in edas.owl), `Delegate` (in iasted.owl), `Listener` (in sigkdd.owl). The heterogeneity of ontologies mainly causes the problem of variation in meaning or ambiguity in entity interpretation and, consequently, it prevents domain knowledge sharing. Therefore, ontology matching becomes a crucial task in semantic web applications.

## 1.2 Challenges in Ontology Matching

According to [28], ontology matching is a process of discovering correspondences (or mappings) between semantically related entities of different ontologies. Because of the high heterogeneity of ontologies, the same concept described in different ontologies may have different representations (e.g., labels, properties or relations with other concepts). Therefore, in the ontology matching process, multiple matching strategies are usually used. In this section, we first illustrate an ontology matching task through a simple example. Then, we discuss the difficulties and challenging issues in ontology matching.

### 1.2.1 Example

Assume that two faculties of computer science in two universities plan to join together and collaborate on some educational projects. The structural organization of each faculty is sketched by an ontology (i.e., faculty1.owl and faculty2.owl in Fig. 1.2 and Fig. 1.3, respectively). In order to conveniently extend opportunities of exchange and collaboration, they should find the equivalent positions in both faculties. It is an ontology matching task.

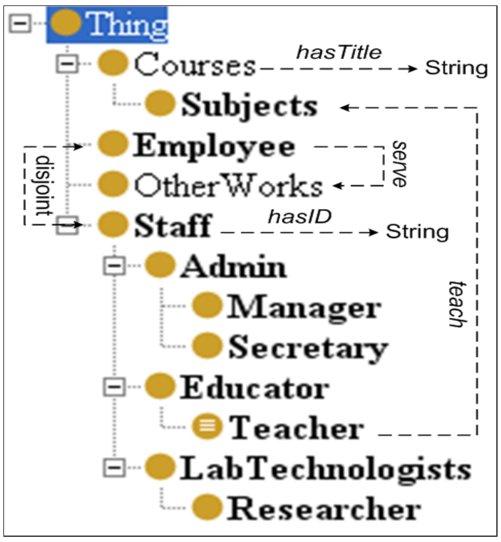


Figure 1.2: Faculty1.owl ontology

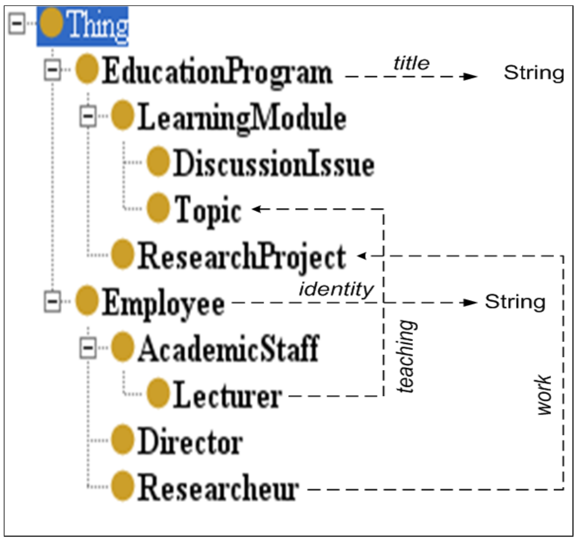


Figure 1.3: Faculty2.owl ontology

At first glance, we can easily find that both ontologies have the same concepts labeled `Employee`. So we can assume that these two concepts are similar. Next, we realize that the label of concept `Researcher` in the **faculty1.owl** is not identical but close to the label of the concept `Researcheur` in the **faculty2.owl**. Indeed, their labels differ in only one character. It may be a typo mistake, so we can assume these two concepts are similar too. Looking at the ontologies again, we see that two pairs of properties such as (`teach`, `teaching`) and (`hasTitle`, `title`) are highly similar because the word "*teaching*" is a variation of the word "*teach*"; the word "*has*" in the label "*hasTitle*" is an auxiliary verb and it can be ignored. In addition, thanks to the dictionary, we find that "*Manager*" is synonym to "*Director*"; "*Subject*" is a synonym of "*Topic*"; thus they are highly similar too. Obviously, the correspondences found at first glance are only based on comparison of the labels of entities in both ontologies. Moreover, finding mappings only by comparing identical labels is not adequate; we should use different techniques in order to discover more results.

Next, note that "*Teacher*" and "*Lecturer*" are not defined as synonyms in the dictionary, thus we need other ways to measure the similarity of these concepts. Because they have similar properties (i.e., `teach` vs. `teaching`) whose value types are similar too (i.e., `Subjects` vs. `Topic`), we can conclude that the two concepts `Teacher` and `Lecturer` are similar. Furthermore, from the structural hierarchy of the two ontologies, we realize that the concept `Staff` in the **faculty1.owl** ontology is similar to the concept `Employee` in the **faculty2.owl** ontology because they have similar descendants, i.e., `Manager` vs. `Director`, `Teacher` vs. `Lecturer` and `Researcher` vs. `Researcheur`. That is, the concept `Employee` in the **faculty2.owl** ontology is matched to two concepts in the **faculty1.owl** ontology, namely `Employee` and `Staff`, which are declared as disjoint. Therefore, the two mappings are incoherent, consequently, one of them is inconsistent and have to be removed. In this case, a mapping between the two concepts `Employee` will be eliminated because they are used in different contexts even though they have the same labels. Indeed, they are a typical example of a polysemy. On the contrary, the mapping between the two concepts `Staff` and `Employee` remains. This mapping is a typical example of a synonym (i.e., the same concept but has different labels).

As it was illustrated in this example, we can see that ontology matching is a difficult and complex process due to the heterogeneity of ontologies. To discover mappings between entities of ontologies, a single and simple matching method will not be sufficient. Instead, it should make use of different methods coming from other related research fields to fulfill this task. For instance, in the illustration example above, we can apply some techniques coming from information retrieval field to discover mappings of entities by their labels. In addition, a method of graph matching could be used to discovers mappings of entities by exploring their structural information. Moreover, to verify the consistent of mappings, a description logic method should be employed.

### 1.2.2 Challenging Issues

Ontology matching can be done either by hand or by using (semi) automated tools. Manual mappings discovery is tedious, error-prone, and impractical due to the high complexity and large scale of ontologies, in terms of their size and their number. Hence, the development of fully or semi automated ontology matching tool becomes crucial to the success of the semantic information systems and applications. In the last decade, through the annual campaign OAEI[1], many ontology matching systems

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

have been proposed. These state of the art approaches have made a significant progress in ontology matching field, but none of them gain a clear success in terms of matching quality performance over all matching scenarios [27].

Several challenges to the field of ontology matching have been outlined in recent research [76]. Among them, *selection of the appropriate similarity measures* as well as *configuration tuning* of their combination are known as fundamental issues that the community should deal with. Indeed, different matching scenarios may require different collections of similarity measures, which consequently require different settings in the combination function. Additionally, *verifying the semantic coherent* of the discovered alignment is also known as a crucial task. Furthermore, the difficulty of the problem grows with the size of the ontologies. Ontology matching in very *large scale* leads to an explosion of the computational space; and consequently, it requires a lot of hardware memory and computational time.

## 1.3 Objectives of the Dissertation

Before proceeding to present the objectives of our research in detail, we clarify the main goal that we will follow in this dissertation. Indeed, the central goal of our research effort is to provide a generic, theoretically sound and practically efficient approach to deal with high semantic heterogeneity in the ontology matching task. Our focus falls on discovering equivalence correspondences between entities at the schema layer of different ontologies. That is, the aim of our approach is to produce mappings on the intentional level (of the kind concept-concept and property-property) and not on the extensional level (e.g., instance-instance) from the input ontologies.

In our approach, an ontology matching system consists of several *matching components* such as terminological matcher, structural matcher, candidate filtering method, and semantic verification method, each of them is used to solve specific issues in the ontology matching task. In particular, in terms of large scale ontology matching, a candidate filtering method first reduces the computational space. Then, terminological heterogeneity of ontologies is solved by a terminological matcher. Next, in order to deal with conceptual heterogeneity, a structural matcher is designed. A combination method then enhances the overall matching result by combining the results obtained by these matchers. Finally, the discovered mappings are refined by a semantic verification method. Besides, a matching strategy is a way that these components cooperate with each other in order to produce an alignment between two ontologies. Therefore, matching strategies and matching components are research objects in our study.

Generally, some matching components can work independently, for example, terminological matcher. On the contrary, the other components( e.g., structural matcher) usually require initial input provided by other matching components. Consequently, their performance strongly depends on the performance of the other ones. Therefore, the performance of the whole matching system is relied on the performance of its components. Based on this observation, our central hypothesis can be stated as follows.

"*The performance of an ontology matching system including matching quality and runtime efficiency can be increased by improving the performance of its components and its matching strategy*".

Therefore, one of the objectives of our research is to improve the effectiveness of the matching components. On the other hand, automatic, flexible and efficient matching strategies (i.e., combination methods) are also fallen in our interest. Explicitly, the following issues have been investigated in this thesis:

1. *How to deal with terminological heterogeneity effectively?*

   Terminological heterogeneity evolves from the same entities having different labels in different ontologies. The aim of a terminological matcher is to discover similar entities by comparing their labels. In practice, many terminological similarity measures have been proposed so far, but none of them could cover all types of the terminological heterogeneity. Therefore, we assume that the combination of different measures may handle this issue. In that way, we propose a method using *Machine Learning* models to combine different similarity measures. It is because machine learning models can self tune their configuration, which is one objective that we focus to.

   On the other hand, unlike the other terminological methods used in existing systems, we assume that the information content of tokens in labels assigned to entities in a domain of interest are not the same. Indeed, some tokens are used more frequently than other, thus they bring less information. This hypothesis leads us to develop a new effective similarity measure based on *Information Retrieval* techniques, which is an alternative method to the machine learning based combination method discussed above when the training data is not available.

2. *How to deal with conceptual heterogeneity effectively?*

   Conceptual heterogeneity evolves from the same entities having different semantic descriptions (e.g., internal and external structural relations, axioms, etc.) in different ontologies. The aim of a structural matcher is to discover similar entities with similar structural information. In order to reduce this type of heterogeneity effectively, we assume that all of the structural relations of entities in the ontology should be taken into account. This evokes a graph matching problem. Therefore, we suggest to make use of an effective *Graph Matching* algorithm for dealing with this issue.

3. *How to effectively combine the matching results of the terminological and structural matchers?*

   Combination of matching results obtained from the terminological and the structural matchers is necessary. This is due to the fact that terminological information and structural information are two independent features of any entity of ontologies. To combine them effectively, we need to know the contribution of these matchers in the matching process. Therefore, the degree of reliability of these matchers should be estimated. Moreover, it should depend only on the ontologies of a given matching scenario.

4. *How to reduce the computational space when dealing with large scale ontology matching?*

   For this issue, we suggest that the reduction of the computational space can be transformed into a *searching* problem. Here, we can apply *Lucene Search Engine* in order to identify the most similar subset of entities in one ontology that possibly match to a given entity in another ontology.

   On the other hand, we observe that if labels of two entities differ in more than three tokens, any string based similarity measures will produce a low similarity value, consequently, these entities are highly unmatched. This observation leads us to an efficient label indexing method to select candidate mappings.

5. *How to detect and remove inconsistent mappings effectively and efficiently, especially in terms of large scale ontology matching?*

   This issue is highly related to the *Ontology Debugging* field. Therefore, we can extend some techniques in this field to semantically refine the discovered mappings.

## 1.4 Contributions

In this section, we explicitly describe our contributions to fulfill the above mentioned objectives.

- Analyze the existing works in the ontology matching field. In this study, we have proposed a survey of related work including a classification of the main contributions in the field.
- New similarity measures to deal with both terminological and conceptual heterogeneity of ontologies. Our experiments show that those methods are better than the existing methods in terms of matching quality.
- A new method based on machine learning models for combining different similarity measures.
- A new combination method that automatically assigns weights to individual matchers. It also automatically determines a threshold value to select the final mappings.
- Effective and efficient filtering methods to deal with large scale ontology matching.
- A method that can detect and remove inconsistent mappings. It is especially efficient in large scale.
- A prototype called YAM++ implementing all the above contributions. This prototype has participated to OARI twice and have got top positions.

The Fig. 1.4 shows the evolution of YAM++ during the research time of this thesis.

We started with YAM++ v.0.0, which was a modification of the previous schema matching tool YAM developed by Fabien Duchateau [23]. The main idea of using machine learning techniques have remained. The main difference is that new similarity measures specializing in the ontology matching field (e.g., wordnet similarity measures, context profile similarity measures) have been implemented and categorized in different groups.

In YAM++ v.1.0, an instance based matching method and a similarity propagation method were added. In order to combine the matching results of those matching methods, a dynamic weighted sum method was added also.

In YAM++ v.1.5, new similarity measures based on information retrieval techniques were added to replace the machine learning method. In addition, a multilingual translator were used to translate labels in different languages into English. Moreover, a semantic verification component were added also.

YAM++ v.2.0 is the current version. This version was designed for performing efficiently large scale ontology matching. For this purpose, new methods for filtering

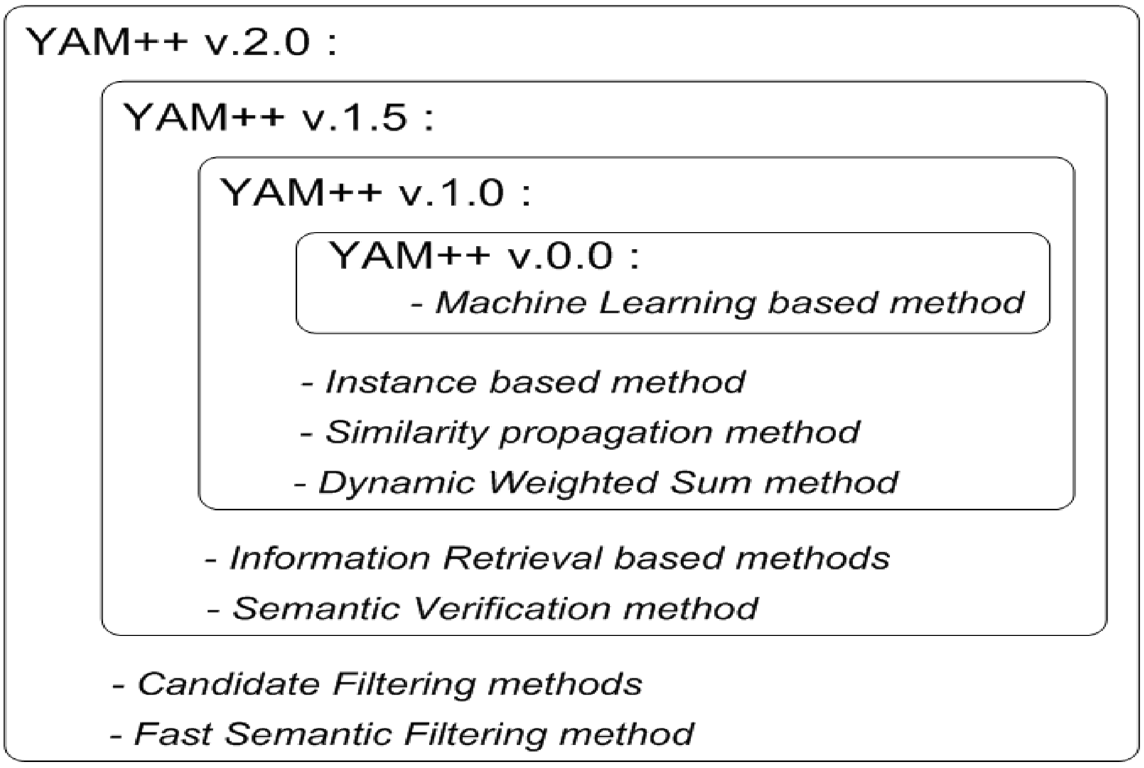

YAM++ v.2.0 :
YAM++ v.1.5 :
YAM++ v.1.0 :
YAM++ v.0.0 :
- Machine Learning based method
- Instance based method
- Similarity propagation method
- Dynamic Weighted Sum method
- Information Retrieval based methods
- Semantic Verification method
- Candidate Filtering methods
- Fast Semantic Filtering method

Figure 1.4: Successive improvements of our matching tools

candidates and a new fast semantic filtering method are added. Besides, a graphical user interface is supplemented in order to help the users debugging the discovered alignment.

YAM++ can be downloaded at: `http://www2.lirmm.fr/~dngo/`

The work on the current PhD thesis have lead to the following publications:

1. DuyHoa Ngo, Zohra Bellahsene. YAM++ : (not) Yet Another Matcher for Ontology Matching Task. (demo paper) EKAW 2012.

2. DuyHoa Ngo, DacThanh Tran, PhanThuan Do. An Information Content Based Partitioning Method For The Anatomical Ontology Matching Task. SoICT 2012.

3. Remi Coletta, E. Castanier, Patrick Valduriez, C. Frisch, DuyHoa Ngo and Zohra Bellahsene. Public Data Integration with WebSmatch. CoRR 2012.

4. DuyHoa Ngo, Zohra Bellahsene, Remi Coletta. A Flexible System for Ontology Matching. CAiSE 2011 - LNBIP 2012.

5. DuyHoa Ngo, Zohra Bellahsene, Remi Coletta. YAM++ – Results for OAEI 2011. In ISWC/OM, 2011.

6. DuyHoa Ngo, Zohra Bellahsene, Remi Coletta. A Generic Approach for Combining Linguistic and Context Profile Metrics in Ontology Matching. In ODBASE, 2011.

7. DuyHoa Ngo, Zohra Bellahsene, Remi Coletta. A Flexible System for Ontology Matching. CAiSE - Forum, 2011.

## 1.5 Outline of the Dissertation

This dissertation is organised into 9 chapters. Current chapter introduced the problem of ontology matching and the objectives of our thesis. In Chapter 2 we present an overview of ontology, ontology heterogeneity and review the existing ontology matching system. In chapter 3, we discuss different types of terminological heterogeneity in detail, and then we propose our methods to deal with them. In Chapter 4, we analyze conceptual heterogeneity and proposed a solution to overcome this challenge. Matcher combination is presented in Chapter 6. In Chapter 7, we discuss existing mapping selection methods and propose our semantic filtering method for large scale ontology matching. Next, in terms of large scale ontology matching, we propose our candidate filtering methods in order to improve the efficiency. The prototypes and evaluation results of YAM++ in OAEI campaigns will be presented in Chapter 8. Finally, conclusion and new perspectives of our research is given in chapter 9.

# Chapter 2

# Literature Overview

In this chapter, we provide a brief summary of the main aspects of the ontology matching field on the basis of the results published so far. The aims of this summary are to give the reader a bird's-eye view on the research topic and to determine our motivations. We start by providing in an informal manner several definitions describing the ontology matching process (section 2.1). Further, we discuss the heterogeneity issues and give an overview of basic matching methods to deal with these issues (section 2.2). Finally, we give a classification of the recent approaches in section 2.3.

## 2.1 Preliminaries

Following Gruber's definition [34], **an ontology** is seen as *a formal, explicit specification of a shared conceptualization in terms of concepts (i.e., classes), properties and relations.* In a more general sense, an ontology can be defined as a collection of concepts and relations holding between these concepts, as well as a set of instances populating these concepts. Ideally, there exists only one ontology describing a domain of interest and it provides a common vocabulary enabling the understanding and sharing of information and data between the members of a given community.

However, as discussed in the introduction, due to the decentralized character of the ontology creation process, it appears in real life that multiple ontologies describe similar or identical domains of the world knowledge - a phenomenon referred to as ontology heterogeneity. We provide an illustration of this heterogeneity in Fig.2.1 showing fragments of two ontologies in OAEI[1] 2009 Benchmark dataset. Ellipses, rounded rectangles, rectangles and dashed rectangles represent classes or datatypes, properties, instances, annotation information or data values respectively. We have

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

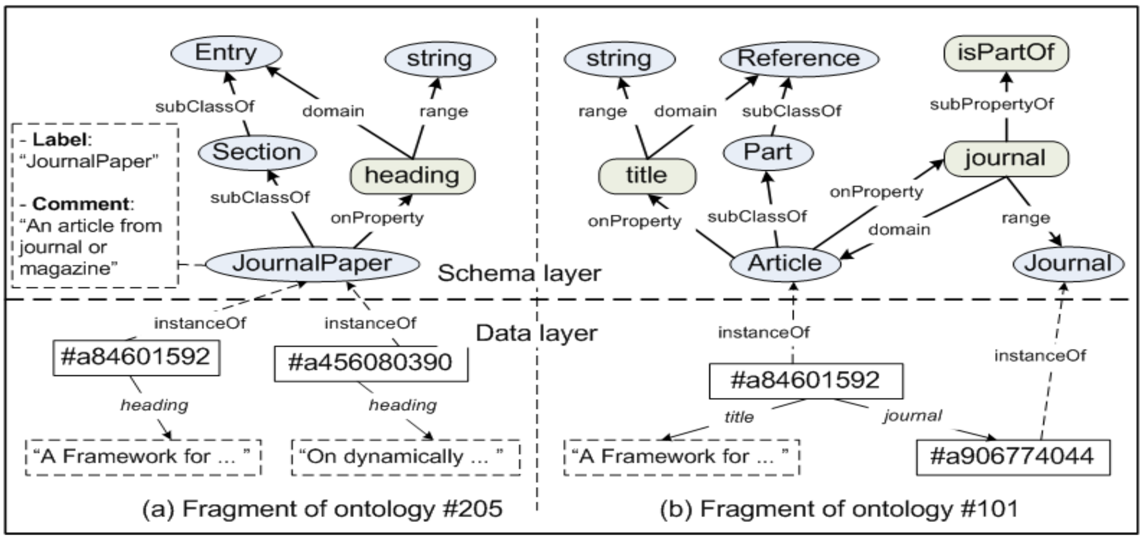


Figure 2.1: Two fragments of ontologies #205 and #101 from OAEI 2009 Benchmark dataset

two layers - a schema layer (concepts, relations) and a data layer (instances). In the schema layer, arrows represent the relations between entities. Dashed arrows from data layer to schema layer indicate that some instances belong to some classes. Arrows in the data layer indicate properties and corresponding data belonging to instances.

The **ontology matching task** aims to bring together two (or more) heterogeneous ontologies by indicating the links between their semantically related elements. An ontology matching procedure applies a **similarity measure** of some kind defined as a function which assigns to a pair of ontology entities a real number between 0.0 and 1.0 expressing the similarity between them [euzenat2007b]. In the example of Fig.2.1, a specific similarity measure known as ScaledLeveinstein[2] applied on the concepts **JournalPaper** and **Article** gives the value 0.17. However, a similarity measure cannot be applied self-dependently - it needs to be a part of a properly designed ontology matching algorithm which takes into account the relations between the concepts, the instances that populate them, possibly background knowledge about the domain of interest, and many other elements relevant to the overall semantic similarity of the ontologies and their components. The result of a properly designed ontology matching procedure ideally is a complete set of semantic correspondences (or mappings) between the entities of two different ontologies [28]. In this thesis, the two notions of "*correspondence*" and "*mapping*" are equivalent.

In the following section, we will discuss in more detail the sources of ontology heterogeneity. This discussion will provide the main axes upon which we will consider

[2]http://sourceforge.net/projects/simmetrics

and explain the existing ontology matching approaches, each dealing with some or several heterogeneity types.

## 2.2 Ontology Heterogeneity

As argued above, understanding the heterogeneity problems and the corresponding solutions is a key to developing a successful ontology matching system. Therefore, we proceed to describe the main heterogeneity types.

### 2.2.1 Sources of Ontology Heterogeneity

The main role of ontology is to describe domain knowledge in a generic, explicit way and to provide an agreed understanding of a domain. Fig. 2.2 describes the ontology engineering process which is the source of ontology heterogeneity. We will go through the different components of this process with regard to the resulting heterogeneities.

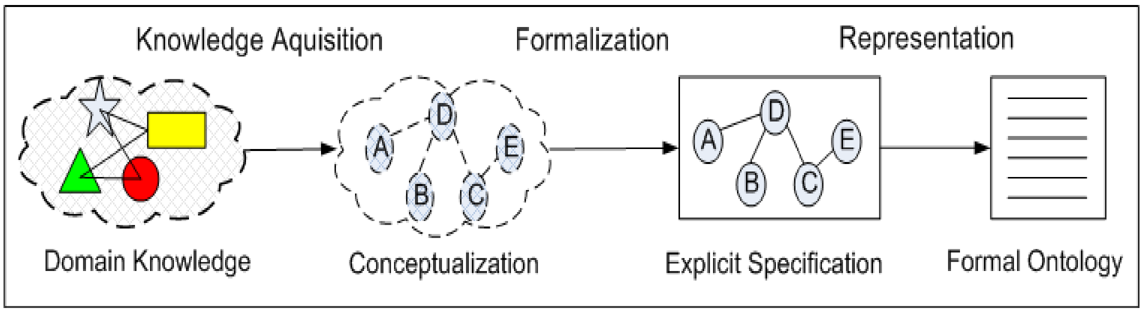


Figure 2.2: Ontology engineering process

The conceptualization process provides a simplified viewpoint of users to the reality (the domain knowledge). However, different users may have different knowledge acquisitions, different backgrounds and understanding of the way this knowledge has to be conceptualized. This leads to different conceptualizations comprising different objects, entities and relations among them.

Further, the formalization process describes all the concepts defined in the previous step as an explicit specification. It means that developers explain all entities and their relations as given in the conceptualization phase, by using some specific formal language (e.g. Description Logics). But, different languages provide different abilities to represent logical rules or axioms and different developers may assign different names for constant or predicates. Therefore, at this step even the same conceptualization maybe formalized with different specification.

Finally, the representation process aims to make those logical definitions and axioms to be formal, and machine-accessible. There exist many kinds of languages can be used in that task such as OWL, KIF, etc. Different represented languages lead to different syntax in the formal ontologies.

From those points above, we see that ontologies are highly heterogeneous. According to the classification of Jérôme Euzenat and Pavel Shvaiko [28], ontology heterogeneity can be classified in four levels:

1. Syntactic: At this level, all forms of heterogeneity depend on the choice of the representation format.

2. Terminological: At this level, all forms of heterogeneity are related to the process of naming entities that occur in an ontology.

3. Conceptual: At this level, all forms of heterogeneity are come from the differences of the content of an ontology.

4. Semiotic & Pragmatic: At this level, all the discrepancies are related to the fact that different individual/communities may interpret the same ontology in different ways in different context.

The heterogeneity at the Syntactic level can be handled by using a transformation tool, which converts ontology from one represented language to another. The heterogeneity at the Semiotic and Pragmatic level are very difficult because they strongly depend on understanding the context of using ontology. Therefore, most of the current ontology matchers focus only on solving problem of mismatches between entities at the Terminology and Conceptual levels. Those types of heterogeneity are also known as semantic heterogeneity. We will see how to overcome semantic heterogeneity in the next section.

### 2.2.2 Overcoming Heterogeneity

An ontology can be created manually, semi-automatically or fully-automatically. However, there are no standard rules widely adopted by all ontology developers. Generally, experts or senior engineers provide the most useful recommendations, conventions in designing ontology process. Following [12, 72], the general framework on building ontology can be viewed in Fig.2.3

There are two parts in ontology design such as: ontology learning and ontology population. Ontology learning aims to describe the interested domain by ontological information (i.e., concepts, properties, relations, axioms) in a so-called "schema

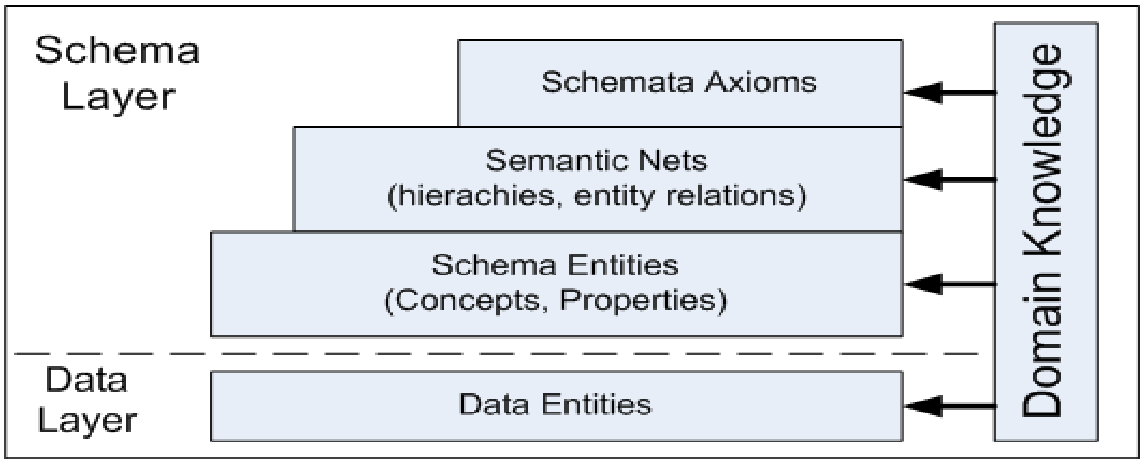

Schema Layer
Schemata Axioms
Semantic Nets (hierachies, entity relations)
Schema Entities (Concepts, Properties)
Data Layer
Data Entities
Domain Knowledge

Figure 2.3: Ontology designing process

layer" (the top part in Fig. 2.3). Ontology population aims to get the extensional aspects of the domain. It finds real instances of the domain that belong to specific concepts and relations mentioned in the ontology learning part. It is also acknowledged as "data layer" (the bottom part in the Fig. 2.3). Let's see a small example of Natural Water Sources ontology [14] in Fig.2.4 in order to understand each level in the framework. In the sequel, we will present and analyze the heterogeneity types with respect to these two parts of the ontology designing process.

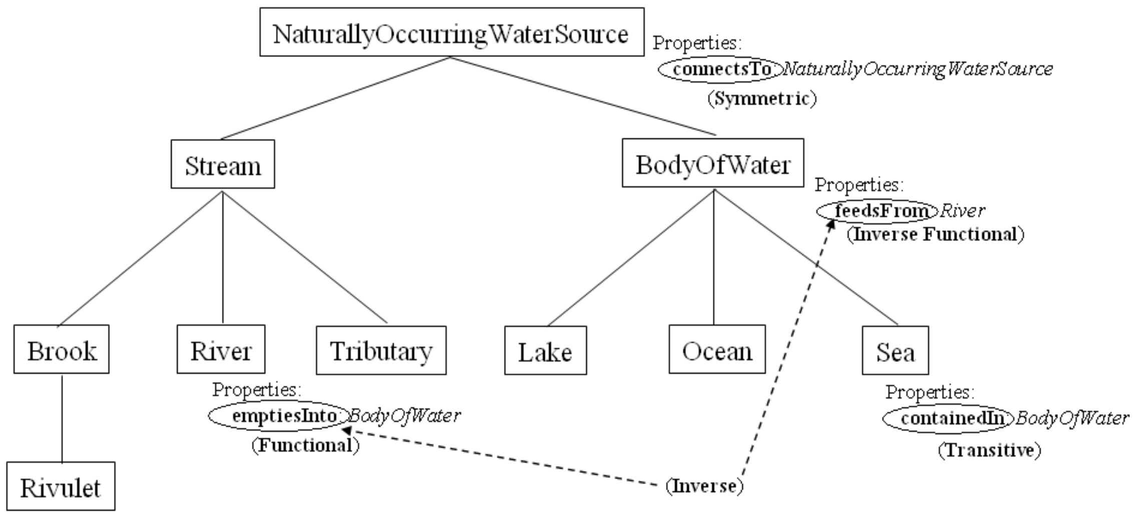


Figure 2.4: An example of ontology fragment

### Schema Entities

The aim of the **Schema Entities** level is to define what concepts and properties are used to describe the user's viewpoint and their understanding on the domain of interest. Generally, at this level, each entitiy on ontology are assigned with an unique identification, some related labels and other annotation, which represent the meaning of entities. For example, rectangles and ovals in Fig. 2.4 represent concepts and properties respectively. Each of them are given a string (e.g., River, Lake, Ocean, Sea, etc.).

According to [12], the first level consists of two sub-tasks corresponding to the **Term** and **Synonym** sub-levels. Most of the Terminological heterogeneity is appeared at this level.

**Terms** are used as signs for entities (concepts, properties). They are linguistic realizations of domain-specific concepts. They convey the implicit meaning by themselves without the ontological consideration. From a linguistic point of view, terms are either single words or multi-word compounds with very specific, possibly technical meaning in a given context or domain [12]. Therefore, if terms are the same or syntactical variation of each other then the entities are probably the same. For example, (*Motion_picture*, *MotionPicture*). In that case, string-based similarity measures can be used to calculate the similarity score between labels. The assumption is that if labels are highly similar, which means the similarity score obtained by a string-based similarity measure on labels are higher than some specific threshold, then the entities probably are the same.

At **Synonym** layer, the task is finding words, terms which denote the same concept in the domain knowledge. For example, (*Thesis*, *Dissertation*) are synonyms in the domain of education. Therefore, if labels are synonym then the entities are probably the same. To deal with that case, we need to use similarity measures, that can exploit information from some external resources (e.g., thesaurus, domain dictionary, etc.) to determine that two labels have the same or close meaning or not. This kind of heterogeneity is known as a semantical variation of terms.

In the study of Mayard and Ananiadou [65], morphology of words is another kind of term variation. For example, (*Match*, *Matches*, *Matching*) may refer to the same concepts. The authors also mentioned that the combination of the three kinds of term variations (i.e., morphological, syntactical and semantical) is very common in naming process for entities. In that case, hybrid methods are frequently adopted. They tokenize labels to tokens, remove stop words, normalize tokens and finally compare tokens by string-based and language-based similarity measures.

Another aspect of Terminological heterogeneity is about using different natural languages to represent labels of entities. For instance, the concept *Booklet* in English and the concept *Livret* in French are the same. To discover matching in that case, multilingual methods are needed. Typically, they use some multilingual dictionaries (e.g., EuroWordnet[3]) or translation system (e.g., Google Translator[4]) or multilingual ontology (e.g., DBPedia[5]) as a common knowledge background. However, one of the main difficulties is that there may be many-to-many translations of words or terms, rather than a single direct correspondence. Therefore, there are many works on ontology matching with a single language, but very few on cross-lingual ontology matching.

### Semantic Nets

The role of the **Semantic Nets** level is to define the hierarchies of concepts and relations between concepts and properties. The main link in the hierarchies is IS-A relation, which produces a backbone of an ontology. The meaning of this type of relation is that the descendant entities inherit all the features of the ancestors. For example, in Fig.2.4, concept *Stream* has property named *connectsTo*, concept *River* is a subclass (IS-A relation) of *Stream*, so concept *River* also has property *connectsTo*. In addition to the concepts and properties hierarchies, the semantic relations between concept-property are provided to specific semantic meaning of those entities in the context of domain. For example, property *emptiesInto* has domain *River* and range *BodyOfWater*. From this definition we can infer that if A has property *emptiesInto* with value B then A is an instance of class *River* and B is an instance of class *BodyOfWater*.

Most of the conceptual heterogeneity and mismatches appear at this level. A classification in detail of different types of conceptual mismatches can be seen in [96, 51]. Most of the conceptualization mismatches are not easily recognized from explicit ontology definition, consequently, it is not easy to deal with such mismatches. The common method for discovering mapping between entities on this heterogeneity level is based on the following intuition: Two entities on two different ontologies are the same if they can be found in the same structural patterns. Here, a structural pattern of an entity involves its internal structure, which is a set of properties, and external structural, which is a set of relationships with other entities.

The idea above can be extended to the similarity propagation idea. It states that if two entities are the same, they propagate some portion of their similarity value to their neighbors in the same relation. Contrary to the patterns above, which are acknowledged as local methods, the propagation idea is widely applied in global methods. In those methods, the similarity propagation is performed over all pair

[3]http://www.illc.uva.nl/EuroWordNet/
[4]http://translate.google.com/
[5]http://dbpedia.org

of entities on ontologies. This process is only stop when the similarity between all pairs of entities reach to stability status. Then, matching pairs of entities are selected according to their final similarity values.

### Schemata Axioms

The highest level in the ontology learning in Fig.2.3 is schemata axioms. By definition, an axiom is a statement expressing a truth in the domain knowledge. In some ontology languages, ontology is considered as a set of axioms including semantic nets. However, in some works (e.g., [12, 24]), the semantic nets and schemata axioms is separated in different levels. The reason is that the semantic nets are considered as backbone of ontology and play a more important role than schemata axioms. The semantic nets exist in almost ontologies, whereas schemata axioms are optional.

Similar to the designing of the semantic nets, different designers with different background knowledge on interested domain may have very different axioms. In general, schemata axioms are used to provide some special characteritics of entities on ontology. For concepts we have *disjointness*, *equivalence*, *restriction* or *cardinality*, etc., and for properties we have *transitivity*, *symmetry*, *functional*, *inverse*, etc. For example, in Fig.2.4, property *connectsTo* is declared in a *symmetry* axiom. This axiom helps us to make an inference like that: if an instance A *connectsTo* other instance B then we can say B also *connectsTo* A. Another example, property *emptiesInto* is declared in a *functional* axiom. It means that if there exist two statements such as: A *emptiesInto* B and A *emptiesInto* C, then we can infer that instances B, C are identical.

Due to the role of schemata axioms on ontology, they are rarely exploited in discovering mappings between entities from two different ontologies. Instead of that, schemata axioms are used to make constraints and to detect inconsistent candidate mappings. For example, consider two candidate mappings (A, B) and (C, D). In one ontology we find that A *equivalent* B, and in another we find C *disjoints* D, so these two candidate mappings are conflict or inconsistent. Users can verify these mappings and remove some of them manually or automatically by using some semantic-based methods.

### Data Entities

As mentioned before, the task within ontology population are thus to learn *instance-of* relation. Here, an instance-of relation is a membership relation between set of data entities (instances) and set of concepts. As same as the upper-levels on schema layer, different designers may assign different numbers of instances for even the same concepts. In practice, many matching systems determine the similarity of two concepts on two different ontologies according to the number of shared data instances. More sophisticatedly, if two instances are matched and they have the same values on two respectively properties then these properties are the same. However, to discover mappings between instances is also a challenge in ontology matching field. Therefore, the ontology matching tasks at schema and instance layers are highly related to each other.

**Domain Knowledge**

In Fig. 2.2, the vertical level - Domain Knowledge is very important. Its role appears in all levels of framework. The intuition is that ontological information describing the same entities is highly similar if ontology designers or developers own the same background of domain knowledge. Therefore, in some specific case (e.g., bio-medical ontology matching), people can use some upper-level ontology or dictionary as common background to discover mappings.

So far, we have presented and analyzed two main types of heterogeneity of ontologies, i.e. terminological and conceptual heterogeneity. Moreover, we have also briefly discussed methods to deal with them in discovering mappings of entities of different ontologies. In the next section, we will present a classification of the existing ontology matching systems in recent years.

## 2.3 Classification of Recent Works

Many ontology matching systems have been proposed to the community so far and several surveys were written on the topic as well. However, most of those surveys [81, 49, 9] had been done before 2006 so they lack the up-to-date information. In the last six years, along with the appearance of the OAEI (Ontology Alignment Evaluation Initiative) campaign, researchers on ontology matching field are able to access to diverse new techniques and solutions. Therefore, the aim of this section is to discuss about recent works that are related to our approach.

In order to make an analysis of matching systems, we view them under the following criteria: (i) basic matching techniques; (ii) efficiency and scalability techniques; (iii) workflow strategies; and (iv) user involvement. These criteria are positioned in 4 layers in Fig. 2.5. Basic matching techniques are the fundamental component of all the matching systems. They are used to discover mappings between entities of to-be-matched ontologies based on some extracted features. Therefore, the basic

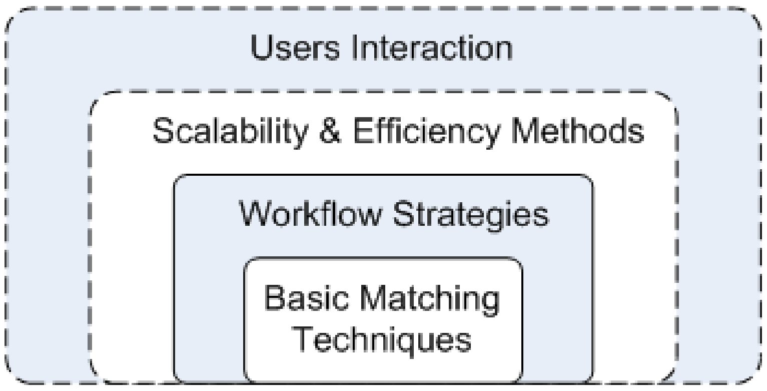


Figure 2.5: Matching layers

matching techniques stand at the lowest layer. On the top of the basic matching techniques is the workflow strategies, which combine different basic matching techniques in a matching process. These two layers are requisite and appear in almost all matching systems. Thus, in Fig. 2.5, their border lines are solid. Many ontology matching systems contain only these two layers.

In recent years, matching large scale ontologies has attracted a lot of interest and attention of the ontology matching community. It requires new efficient techniques to deal with huge number of pairs of entities in the computational space. Due to the difficulties of the large scale ontology matching, many systems lack of these techniques. Therefore, in Fig. 2.5, the scalability and efficiency methods appear on top of the workflow strategies layer and they are remarked by a dash border line. Finally, the mapping result should be verified by the user or the domain experts. They can also interact in some phases of a matching process in order to produce a better matching quality. That is, its layer is stood on the top of Fig. 2.5.

### 2.3.1 Basic Matching Techniques

The basic matching techniques are also known as basic or individual matchers [81] because they usually are an implementation of a similarity measure working on specific feature of entities; they can work independently and gain only a partial mappings. That is why in almost all matching systems, several basic matchers are combined by some strategy in order to potentially compensate for the weakness of each other. According to the survey in [28] and descriptions of participant tools on OAEI campaign, the basic techniques can be divided in the following categories.

**Terminological methods**. These methods compare strings encoded in the names, labels or comments of entities to find those which are similar. They include string-based, language-based and combined string-language based methods. These

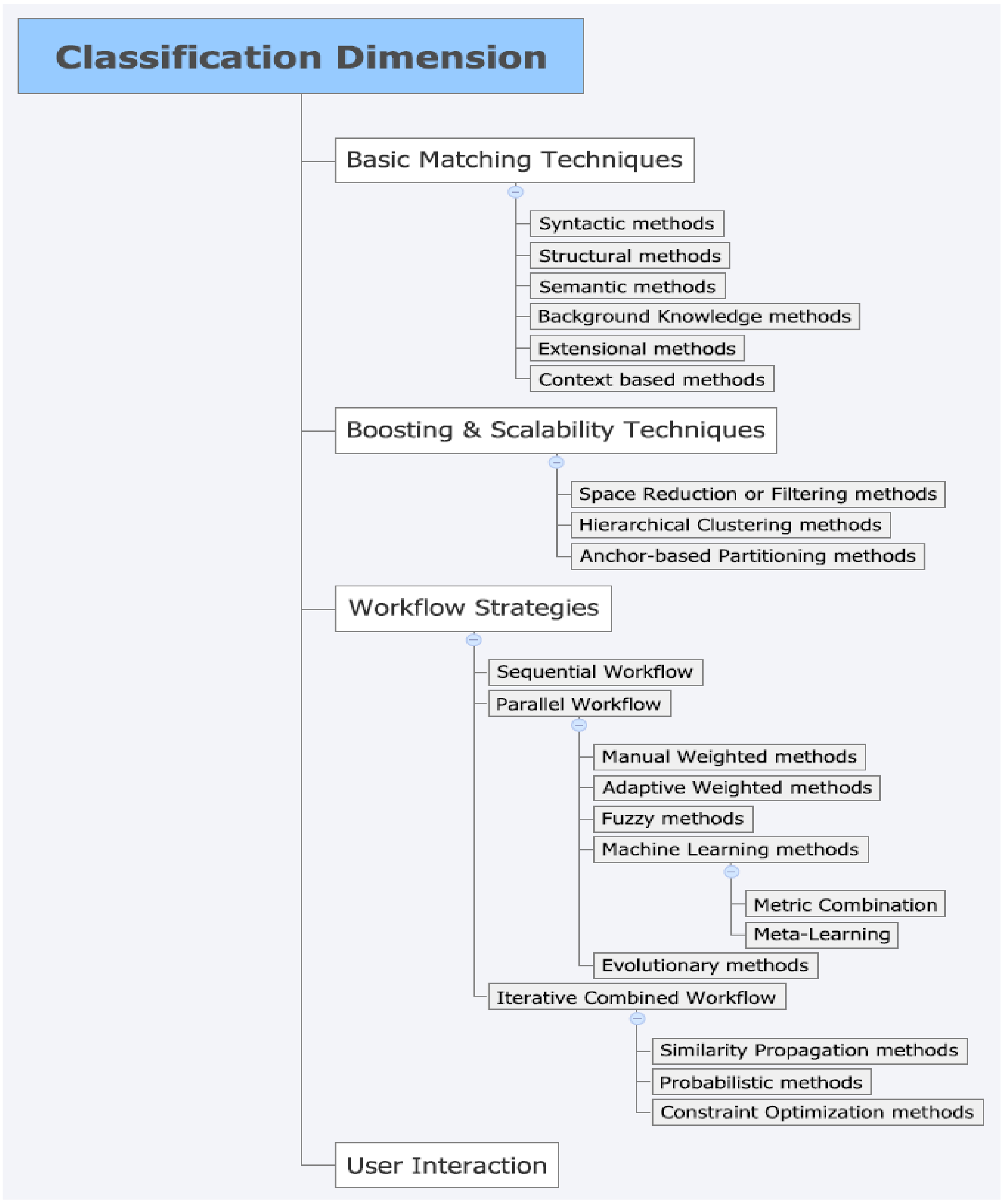


Figure 2.6: A Classification of matching systems

methods can be found in every tools (e.g. COMA [4], AgreementMaker [26], ASMOV [42], etc.) and their mapping results are normally used as input to other methods.

**Structural methods**. These methods exploit the structural information of entities of ontologies to find mappings. The structural information here is the relationship between entities such as sub-classes, domain, range, restriction property, etc. According to survey in [28], they are divided in two sub types such as internal and external structure methods. Internal structure of entities use criteria such as domain, range and cardinality of their properties to calculate the similarity between them. A typical example of using these methods are ASMOV [42], COMA [4]. However, these relation are generally weak (for example, many classes are restricted by the same properties or many properties have the same domain, range), so these kinds are commonly referred as constraint based approaches in order to remove wrong mappings rather than to discover accurate correspondences between entities. This way is successfully applied in GLUE [21], PRIOR++ [64], CODI [41].

External structure methods compare entities based on their position on the hierarchies of ontologies. Many systems (e.g. COMA[4], ASMOV [42]) exploit this idea to develop their structural similarity metric. Commonly, these metrics calculate how much overlap between two set of entities taken from super entities or descendant entities or on the path from the root to the to-be-matched entities. However, the structure of ontologies strongly depends on the viewpoint of designers on a knowledge domain, so there is no guarantee that the position and neighbors of the same entities are the similar. Therefore, similar to internal methods, some systems use external information to filter incorrect mappings (e.g. AgreementMaker [26]).

**Extensional methods**. As the information at data layer can give important insight into the contents and meaning of entities at schema layer, this kind of methods compare entities based on their set of instances. For example, similarity between two classes can be calculated by how much percent they shared the same instances; or two properties are consider as match if they are assigned by the same values in two similar instances. Besides, instances data can be used to build some probabilistic methods to compute how much two entities are related. These methods are very useful when schema information is limited. They can be found in many systems such as ASMOV [42], AROMA [17], GLUE [21].

**Semantic methods**. The main characteristics of these methods is using semantic information, which encoded by the description logic, to detect inconsistent mappings. Based on the description logic, some reasoners can be used to infer the implicit relation to be explicit. For example, KOSIMAP [83] use Pellet[6] to rewrite

[6]http://clarkparsia.com/pellet/

and expand relations between entities. Other type of using description logic is to transform the selected mappings task to the optimization problem on constraint programming. That way can be found in S-Match [33] with propositional satisfiability SAT solver[7], CODI [41] with Markov Logic Network, GLUE [21] with label relaxation approach.

**Background Knowledge methods**. Correctly understanding the meaning of the concepts not only reduces misunderstanding but also discovers high accurate mappings between entities of heterogeneous ontologies. Along with the development of semantic web, various ontologies from the same domain or different domains may be integrated to an upper level ontology. For example, UMLS[8] is a generic medical dictionary; DBPedia[9], YAGO[10] are multi-purpose upper ontology; Uberon, FMA, GO are specific upper level for bio-medical domain[11]. Using those upper level ontologies is common background knowledge to obtain the definition for each concept. Recently, many matching tools utilize these techniques to discover mappings. For example, CODI [41], ASMOV [42] exploit UMLS; AggrementMaker [26] uses Uberon; BLOOMS [78], LDOA [48] uses Linked data in DBPedia.

**Context-based methods**. This kind of methods is somehow a mix of terminological and structural methods. The main idea is that two concepts are similar if they are found in similar contexts. This idea is inspired from the information retrieval field. To do that, for each concept on ontology, a context profile is built by gathering description of its neighbor such as parents, children and properties. The context profile is also called by name Virtual Document [80]. A vector space model is then constructed from those documents in order to assign weight to each term in it. Similarity score between two concepts is computed by how much their virtual documents are similar. The context-based methods can be found in Falcon-AO [39], RiMOM [58], Prior++ [64], AgreementMaker [26], MaasMatch [84]

### 2.3.2 Workflow Strategies

Many basic matching techniques have been proposed so far, but none of them can fulfill the requirement of users 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 put several basic techniques (or basic/individual matcher) in work are needed. According to [28], there are

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

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

[9]http://dbpedia.org/

[10]http://www.mpi-inf.mpg.de/yago-naga/yago/

[11]http://bioportal.bioontology.org/

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 for improving the efficiency and quality matching results. Their combination strategies are acknowledged as hybrid composition because each phase in their workflow can be applied one of the basic types. In this section, we discuss about 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 semantical matching methods require predefined mappings. To do that, an element-level method first produces initial mappings and then passes them to the structural method. This type of strategy can be found in CUPID [59], Falcon AO [39], Similarity Flooding [67], where a terminological metric is the first matcher and produce initial mappings to the second matcher (structural matcher); or in CODI [41], PRIOR++ [64] the first matcher aims to find all possible candidate mappings, then in the next phase, a semantic matcher refines them by eliminating inconsistent mappings.

**Parallel Workflow**. In this strategy, all 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, systems may be implemented with manual assign weights (COMA [4], ASCO [56]) or with adaptive weights (the first phase in PRIOR++ [64], AgreementMaker [26]) or fuzzy assign weights (OWA [44]). Besides, aggregation operators can be seen as a decision functions which decide two entities are matched or not. In that way, 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 on entities. Example systems of this kind of combination are [75, 68]. Similarly, belief theory Dempster-Shafer framework DDSim [70] and evolutionary methods MapPSO-MapEVO [7] are also used as aggregation operators.

**Iterative Workflow**. The idea of this strategy is that the matching process runs repeatedly several times until meets the stop condition. The matching process can be performed by one basic matcher or combination of several basic matchers. This type of strategy can be used in a part of system or in the whole system. A typical example of using iterative mwthod in a part of system is 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 running algorithm, similarity value of each pair of entities is re-computed according to the current values of itself and its neighbors. We can find this strategy in the second phase of works in SimilarityFlooding [67], OLA [30, 29], Falcon-AO [39], Lily [97], RiMOM [58], AgreementMaker [26]. Similarly, 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 mappings or removes inconsistent mappings. This process will be stopped until the optimization condition is reached. The iterative constraint-based methods can be found in second phase of GLUE [21], CODI [41], LogMap [45].

Typical examples of using iterative in the whole system can be found in ASMOV [42], QOM [25]. In those systems, the matching process consists of several basic matching techniques. For example, in the first phase of ASMOV, they use parallel composition for terminological, structural and extensional metrics, then pass the discovered mappings to the second phase to run semantic verification process. Those two phases are repeated several times until there is no change found in the discovered mappings.

## 2.3.3 Efficiency and Scalability Techniques

High quality is the prime importance that needs to consider in development of all matching tool. Besides, efficiency of matcher is also very important, especially, when the input ontologies are too big whereas memory 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 [10], 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 sophisticatedly, in QOM [25], the heuristics strategies based on different extracted features such as label, hierarchy, neighours, etc., are used in each iteration to select the promising mappings.

**Partitioning methods**. In this category of approaches, two large ontologies are firstly slitted 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, some heuristic techniques are proposed. Based on those heuristics, instead of all pair of sub-ontologies, only the high relevant pairs of sub-ontologies will be passed to the matching process. These methods can be found in Falcon-AO [39] or COMA++ [4].

A sub-class of this category is known as **Anchor-based Partitioning** methods. These methods are modification of the algorithms above, which partition to-be-matched ontologies according to the set of anchors. In short, an anchor is a pair of entities mapping determined by a similarity metric. A fragment or sub-ontology is constructed by collecting neighbors entities of the chosen anchors across ontologies. At the end of partition phase, each sub-ontology of the source ontology will have a corresponding sub-ontology of the target ontology. Then, the alignment process will be performed for each pair of related sub-ontologies. These methods can be found in Anchor-Promt [72], AnchorFlood [38], Lily [97], TaxoMap [37].

### 2.3.4 User Involvement

Most of the current matching systems aim to full automated discover alignment between entities from different ontologies. However, despite of much effort of researcher and developers, there is no system or tool can obtain 100 percents of matching quality. In fact, matching tools or systems only provide a list of candidate (with high confidence) mappings, but the final results are up to the users decision, in which correct mappings will be selected and incorrect ones will be deleted. Users involvement can be seen in the whole process including: pre-match, match and post-match.

At the **pre-matching phase**, users can recommend relevant background knowledge in advance (for example, ASMOV [42] uses UMLS; AgrrementMaker [26] uses upper-ontologies such as Uberon, FMA for ontology matching in bio-medical field). Users can also provide predefined weights for each matching component, or they can suggest aggregation function to combine different individual matchers. For example, AggreementMaker [26], COMA++ [4] allow users select different configuration for the system.

At the **matching phase**, users can interact with system in order to give the feedback (for example, suggestion to remove some inconsistent mappings) after each iteration. The matching system then updates the suggestion to next iterations. This type of involvement can be found in PROMPT [74].

At the **post-matching phase**, users validate the candidate mappings in order to remove incorrect ones if a matching system provides a graphical interface, in which ontologies and all discovered mappings are displayed on the screen. According to the users background knowledge on the matching domain, they can verify found mappings are correct or not. There are several matching systems supporting very nice interface such as COMA++ [4], AgreementMaker [26].

## 2.4 Conclusion

In this chapter, we have presented the main notions of ontology matching field used in this thesis. Afterward, we have briefly reviewed the four types of heterogeneity of ontologies. They consist of syntactic, terminological, conceptual and semiotic & pragmatic heterogeneity. We have argued that the main focus of the ontology matching task is to overcome the challenging issues caused by terminological and conceptual heterogeneity (Section 2.2).

In Section 2.3, we have classified and reviewed the state of the art ontology matching systems through four criteria: basic matching techniques, efficiency and scalability techniques, workflow strategies, and user involvement. Thanks to this classification, we have obtained an overview of the main aspects and characteristics of the existing ontology matching systems.

Henceforth, in the following chapters of this thesis, we are going to answer to the list of research issues initiated in the Introduction chapter.

# Chapter 3

# Dealing with Terminological Heterogeneity

Terminologies play an important role in the ontological representation of domain knowledge. They are collections of symbols which need to be interpreted as evoking some concepts as well as referring to some concrete objects in the real world. The connections between objects, concepts and symbols are shown in the so called meaning triangle of Ogden and Richards [47]. For example, a string "*Leopard*" evokes and furthermore denotes a specific concept of a large animal of the cat family. From this point of view, a high possibility is that the same concepts are signed by the same labels or highly similar labels in the same context of domain knowledge, provided that the same natural language is used.

However, in reality, labels of the same concepts may be highly different because of different conventions in naming process. Note that, naming is the process of associating a linguistic object from a public language (i.e. strings or sequence of characters) to entities described in an ontology. In fact, because there is no standard rule for labeling concepts in an ontology, the same concepts in the same domain of knowledge might be assigned with different labels in different ontologies created by different ontology developers. These differences define various challenges for the ontology matching community since discovering mappings of entities by checking only identical labels is not sufficient. Therefore, effective similarity measures are needed to overcome this challenge.

In this chapter, we discuss our approaches to deal with the terminological heterogeneity issue in discovering mappings between ontologies. We first study different types of terminological heterogeneity of ontologies in section 3.1. The analyses of these types could help us understand more about the diversity of terminological heterogeneity and understand more about the difficulties to deal with these problems.

Next, in section 3.2, we classify the existing similarity measures that are widely used in state-of-the-art ontology and schema matching systems. The classification is based on the main characteristics of these measures. In addition, we discuss the capacities of these measures in terms of strengths and weaknesses to deal with the different types of terminological heterogeneity mentioned above.

The main contribution of this chapter is the definition of three advanced similarity measures which will be described in detail in section 3.3. They include an Information retrieval-based similarity measure (section 3.3.1), a Context profile-based similarity measure (section 3.3.2) and an Instance-based similarity measure (section 3.3.3).

Finally, the experiments show the strength and weakness of our similarity measures in section 3.4.

## 3.1 Analyses of the Terminological Heterogeneity

In this section, different types of terminological heterogeneity are analyzed in order to have a full picture about this challenge. According to our study of the terminological information of different ontologies, we categorize the terminological heterogeneity of labels into six main types. Before discussing these types in detail, we define three aspects of matching strings:

- Two strings are **syntactic similar** if they are identical or they differ in only few characters at few positions.
- Two strings are **meaning similar** if they are synonym or very close according to their definition in a thesaurus (e.g., Roget's Thesaurus) or in a lexical database( e.g., Wordnet).
- A string is **tokenizable** if there are split signs between their tokens such as: upper-lower cases (e.g., "*MasterThesis*", "*conferenceMember*", etc.), special character like underline, whitespace, punctuation or digit number, etc., (e.g., "*phd.thesis*", "*Conference_paper*")

According to these aspects, the six terminological heterogeneity types are categorized as follows:

- **Type 1**: Two labels are syntactic similar. It commonly happens when one label uses special characters to separate its tokens, whereas, the other does not. For example, "*issn*" vs. "*I.S.S.N*", "*firstname*" vs. "*First_Name*", "*email*" vs.

"*E-mail*", "*Ph.d_Student*" vs. "*phdStudent*" etc. Besides, this case might be happened when there is a typo in one label or by using multiple instead of singular. For example, "*sponsor*" vs. "*sponzor*", "*researcher*" vs. "*researcheur*", "*pccard*" vs. "*PCcards*", etc.

- **Type 2**: Two labels are not syntactic similar but they are tokenizable and their tokens are one by one syntactic similar. For example, "*PC_Chair*" vs. "*Chair_PC*", "*SubmissionsDealine*" vs. "*deadline_submission*", etc.

- **Type 3**: Two labels are not syntactic similar but they are meaning similar, for example, "*booklet*" vs. "*brochure*", "*compilation*" vs. "*collection*", etc.; or they are tokenizable and their tokens are one by one syntactic or meaning similar, for example, "*Conference_participant*" vs. "*conference-attendee*", "*ConferenceDinner*" vs. "*Conference_Banquet*" etc.; or few of tokens are stop words, for example, "*has_the_last_name*" vs. "*hasLastName*", "*email*" vs. "*has_an_email*", etc.

- **Type 4**: Two labels are not syntactic similar and only a part of them (one or few tokens) is syntactic similar or meaning similar. For example, "*Document*" vs. "*Conference_Document*", "*Co-author*" vs. "*Contribution_co-author*", "*Attendee*" vs. "*Conference_participant*", etc.

- **Type 5**: Two labels are totally different in terms of syntactic or in terms of meaning. In this case, maybe one label is an acronym or an abbreviation of the other (e.g., "*WWW*" vs. "*WorldWideWeb*" vs. "*Website*", "*Misc.*" vs. "*Miscellaneous*") or maybe they are only closely related in a specific domain (e.g. "*Contribution*" vs. "*Paper*", "*Information_for_participants*" vs. "*Programme_Brochure*", etc., in the Conference organization domain).

- **Type 6**: Labels are represented by different languages. For example, a label is written in French (e.g. "*livre*" and the other is written in English (e.g. "*booklet*").

In fact, the type 6 of heterogeneity can be transformed to one of five types above when labels are translated into the same language (e.g. English). Therefore, we mainly focus to find solutions for the first five types of heterogeneity. Now, we are going to review the existing similarity measures and propose a new similarity measures to deal with these heterogeneity types discussed above.

## 3.2 Basic Terminological Similarity Measures

A lot of terminological measures have been proposed so far to apply in different scientific fields like record linkage and record matching, data duplication in database [69, 99], entity or object identification in information retrieval [94], discovering common molecular subsequences in bioinformatic [87], etc. According to [28], terminological-based similarity measures can be divided into two main groups like string-based and language-based similarity.

### 3.2.1 String-based Similarity Measures

A similarity score between two strings computed by a string-based similarity measure depends on their sequences of characters only. Following to the comprehensive survey in [13], string-based similarity measures can be categorized into three following types: edit-based, token-based and hybrid.

#### Edit-based Similarity

In general, edit-based similarity value of two strings is computed by counting different edit operations to transform one string into the other, e.g., insertion of characters, character swaps, deletion of characters, or replacement of characters. Depending on the computation method, the two following families of edit-based similarity measures are frequently used in practice:

- **Edit distance** family. Here, the edit distance between two strings $s_1$ and $s_2$ is the minimum cost of transforming $s_1$ into $s_2$ using a specified set of edit operations with associated cost functions. For example, the **Levenstein** method assigns the cost of all operations to 1, the **Hamming** method only assigns the cost of replacement operation to 1, whereas, **SmithWaterman**, **Needleman-Wunch**, **Gotoh** methods assign different costs to different operation types.

- **Jaro distance** family. The similarity computation of similarity score in this family is mainly based on the number and the order of the common characters between two strings $s_1$ and $s_2$ (like **LCS** - longest common substring method). **Jaro** method also takes the cost of transpositions of common characters where a transposition occurs when the i-th common character of $s_1$ is not equal to the i-th common character of $s_2$ in computation. An extension of the **Jaro** similarity, called the **Jaro-Winkler** similarity, emphasizes the importance of common prefix between $s_1$ and $s_2$. Recently, a new extension ISUB [90] takes into account not only the similar part but also the different part of two strings.

According to its experimental results, ISUB seems to perform well in ontology matching field.

In practice, the edit distance value of two strings is commonly transformed to their similarity value by counting the compensation of its normalized value. For example, $ScaledLevenstein(s_1, s_2) = 1 - \frac{Levenstein(s_1,s_2)}{max(size(s_1),size(s_2))}$.

### Token-based Similarity

Token-based similarity measures compute the similarity score between two strings by comparing their tokens rather than the whole string themselves. It means that they first tokenize the input strings $s_1$ and $s_2$ into two collections of tokens. Intuitively, tokens correspond to substrings of the original string. For example, `tokenize`("*ConferenceParticipant*") is {"*Conference*", "*participant*"}, here `tokenize`() is a tokenization function. Then, the similarity score between two original strings is computed by measuring the degree of similarity between their collections of tokens.

In fact, there are three factors impacting to the similarity computation of token-based similarity measures.

- The first factor is related to tokenization method. Most of the token-based similarity measures rely on special signs between tokens in a string to perform tokenization. In this case, the input string is tokenizable. Besides, even if the string is not tokenizable, it can be split by a Ngrams methods, in which each token is a chunk of characters with fixed size N.

- The second factor is related to internal similarity measure that computes the similarity scores between tokens. The frequently used internal measure is an **Identical** measure, which returns 1.0 if two tokens are absolutely equal, otherwise 0.0. Several token-based similarity measures using the **Identical** measure are: **Ngrams**, **Manhattan** distance, **Euclidean** distance, **Jaccard** similarity, **Overlap**or **Dice** coefficient, **TFxIDF** measure, **Cosine** similarity, **Jensen-Shannon** divergence, etc. However, the strict equal condition faces a problem when tokens are not identical but syntactically similar. For example, the Identical measure discovers only the first pair of tokens but not the second in two strings "*ConferenceMember*" and "*Conference_Members*". To solve this problem, the similarity score between tokens can be computed by applying an edit based similarity measure. There are examples of token-based measures: **Monge-Eklan**, **Level2**, **ExtendedJaccard**, **SoftTFIDF**, etc.

- The last factor is related to a computation similarity of two collections of tokens. It may use an operation on set, in which all tokens have the same weight (e.g., **Overlap**, **Dice** coefficient, **Jaccard** similarity); or an arithmetical function with setting weights to tokens (e.g., **TFxIDF**, **SoftTFIDF** measure, **Cosine** similarity); or a probabilistic function (e.g. **Jensen-Shannon** divergence, **Fellegi-Sunter** method).

### Observations on String-based Similarity Measures

After doing the analysis on both edit-based and token-based similarity measures, our observation of the relation between these measures and different types of terminological heterogeneity is as follows:

- The main advantage of token-based similarity measures is that the similarity is less sensitive to word swaps compared to similarity measures that consider a string as a whole (notably edit-based measures). For example, "*MemberConference*" and "*Conference_Member*" are similar. To overcome typographical errors or syntactically similar tokens, a token-based similarity measure must use an edit based measure as its internal measure. Therefore, token-based similarity measures can be used to deal with type 2 of terminological heterogeneity.

- The edit-based similarity measures and Ngrams measure are more appropriate than token-based measures when one of two strings is not tokenizable. Besides, edit-based measures and Ngrams measure are less impacted by typographical errors than the basic token-based measures. Therefore, these measures can be used to deal with type 1 of terminological heterogeneity.

- Several measures like **TFIDF**, **SoftTFIDF**, **Jensen-Shannon** and **Fellegi-Sunter** [13] do need an external resource to assign weights to tokens. They face to a limitation that a large corpus of text related to a given domain may not be available. Therefore, these measures are mainly used in object identification within a large database.

- None of these measures can deal with type 3 of terminological heterogeneity. It is because they compute the similarity score by using syntactic feature only. To overcome this type of heterogeneity, we propose a combination of string-based and language-based similarity measures to compute the similarity score between tokens.

### 3.2.2 Language-based Similarity Measures

Language-based measures focus on the meaning and grammatical form of words to find the association between them. Many techniques used in these measures are inherited from the Natural Language Processing field. They can be divided into two intrinsic and extrinsic measures.

#### Intrinsic Similarity Measures

The intrinsic similarity measures are based on using the internal linguistic properties of words to perform the terminological matching. The key idea of the intrinsic measures is to perform term normalization with the help of morphological and syntactic analysis. In the real world, morphological variants are one of the main reasons making the terminological heterogeneity of ontologies. For example, "*matching*", "*matched*" and "*matches*" are three variations of the term "*match*". In order to perform a normalization of a term, we can use a *stemming* algorithm (e.g., Porter stemmer [79]), which strips words to their root form by removing suffixes such as plural forms and affixes denoting declension or conjugation. However, term variations are not only about the difference of suffixes or affixes, for example, "*bring - brought*", "*leaf - leaves*", etc. In that cases, because these words are irregular, we need to use a dictionary (e.g. Wordnet) to find the base form of each word.

#### Extrinsic Similarity Measures

The extrinsic similarity measures make use of external knowledge resources such as dictionaries, lexicons or thesauri. They mainly rely on the semantic relationships in the dictionary hierarchy to compute a similarity score between words. For example, they can find that "*Attendee*" is synonym with "*Participant*", "*Facial*" is an adjective and related to noun word "*Face*", etc. In that case, language-based similarity measures easily conclude that these words are similar and assign value 1.0 to their similarity scores. However, if words are not synonym, for example "*Book*" is hypernym of "*Booklet*", etc., a computation function is needed to determine how much these words are similar.

In practice, Wordnet lexical database is widely used to compute the semantic similarity between English words. In terms of multi-languages matching, a multilingual dictionary or a multilingual translator are necessary to convert labels into the same language. Generally, similarity measures designed for the Wordnet can be used with other semantic network resources (e.g., MeSH (Medical Subject Headings)). According to survey [8], Wordnet-based similarity measures can be divided into two main types:

- Edge-based similairy measures compute the similarity of words by the distances of the positions of their senses in the Wordnet hierarchy. The intuition is that the shorter the path from one sense to another, the more similar they are. Typical examples are **Path**, **WuPalmer**, **LeacockChodorow**.

- Information-based and integrated similarity measures compute the similarity of words through the position of their senses in the Wordnet hierarchy and their information content. Here, the information content (IC) of a concept provides an estimation of its degree of generality or concreteness, which enables a better understanding the semantic meaning of the concept. This idea of using information content in computing semantic similarity of words was first proposed by Resnik, then it have been extended in other measures like **Lin**, **JiangCorath**, **Seco**, etc.

Besides, like techniques used in Natural Language Processing and Information Retrieval fields, the extrinsic measures ignore stop words in their similarity computation. Note that, stop words are the common words which would appear to be of little value in labels. For example, in English, articles "*a, an, the*" or preposition "*on, in, at*", etc., are stop words. Then, the key word in label "*has_an_email*" is "*email*". Furthermore, for other languages, there exist others stop word lists.

### Observations on Language-based Similarity Measures

Obviously, to compute the semantic similarity of words, the extrinsic similarity measures are more important than the intrinsic similarity measures. In fact, the intrinsic measures mainly deal with the syntactic variations of words but do not deal with their meaning. For example, the intrinsic measures can normalize terms "*booklets*" to "*booklet*", but they cannot show the semantic relation between "*booklet*" and "*brochure*". Therefore, the intrinsic measures cannot recognize that "*booklets*" and "*brochure*" are similar. Whereas, the extrinsic measures with the help of a complete dictionary (e.g., Wordnet) can find the basic forms and all possible senses of words to compute the similarity score between them. For example, the extrinsic measures find the basic form of "*booklets*" is a noun word "*booklet*" and it is synonym with noun word "*brochure*".

However, both of the intrinsic and extrinsic measures may suffer from the two following problems. The first problem is when the input words cannot be found in the dictionary. It can be caused by typographical errors (e.g., "*sponsorship vs.*

*sponzorship*", etc.) or by the words themselves. The second problem is that both intrinsic and extrinsic measures mainly deal with single words but not compound words, which are frequently assigned to labels of entities in ontologies. For example, neither of labels "*DoctoralThesis*" or "*PhdDissertation*" are found in Wordnet. But each of their token can be found in Wordnet and they are one by one closely related (i.e., "*Doctoral vs. Phd*" and "*Thesis vs. Dissertation*".

Therefore, like the observation in Section 3.2.1, a combination of string-based and language-based measures is needed to compute the similarity score of two words (or tokens) in general. Furthermore, in order to deal with type 3 of terminological heterogeneity, we should use this combination as an internal similarity measure for the token-based similarity measures.

### 3.2.3 Hybrid Similarity Measures

Now we present hybrid similarity measures, which are a combination of string-based and language-based similarity measures, in order to deal with type 3 of terminological heterogeneity. In particular, we split a hybrid measure into two parts: (i) compute a similarity score between tokens and, (ii) compute similarity score between compound tokens.

#### Combination of a String-based and a Language-based Similarity Measures

The main idea is as follows. Firstly, a morphological method finds all possible basic forms of each token in the dictionary (e.g., Wordnet). If the basic forms of both tokens exist, an extrinsic-based similarity measure is used to compute the similarity score between every pair of basic forms of tokens. Otherwise, the similarity score of two tokens is computed by a string-based measure.

This idea is presented in the Algorithm.1. Here, function `MorphologicalForms` takes a token as input and finds all possible senses and morphological forms of token existing in Wordnet dictionary. For example, let's compare "*teach*" and "*teaching*". `MorphologicalForms`("*teach*") returns { noun: "*teach*" , verb: "*teach*"}; `MorphologicalForms`("*teaching*") returns { noun: "*teaching*", verb: "*teach*"}. Because two obtained sets of senses have a common {verb: "*teach*"}, therefore token "*teach*" and token "*teaching*" are similar. Another example, let's compare "*sponsorship*" and "*sponzorship*". Because token "*sponzorship*" cannot be found in Wordnet, $MF_2$ is empty. If a string-based measure is **ScaledLevenstein**, then the similarity score of two tokens is equal to `0.91`.

**Algorithm 1:** COMPUTE SIMILARITY BETWEEN TWO TOKENS - $sim(token_1, token_2)$

```
input  : $token_1, token_2$ : two tokens,
         $dictMetric$ : a language-based similarity measure,
         $stringMetric$ : a string-based similarity measure,
         $pos$ : part of speech function
output: $score$ : a numerical value
1 $MF_1 \leftarrow \texttt{MorphologicalForms}(token_1)$
2 $MF_2 \leftarrow \texttt{MorphologicalForms}(token_2)$
3 if $(MF_1 \neq \emptyset) \wedge (MF_2 \neq \emptyset)$ then
4     $score \leftarrow \max_{t_i \in MF_1, t_j \in MF_2}(\texttt{dictMetric}(t_i, t_j))$
5     where   $\texttt{pos}(t_i) = \texttt{pos}(t_j)$
6 else
7     $score \leftarrow \texttt{stringMetric}(token_1, token_2)$
8 end
```

### Computing Similarity Value between Compound Labels

The idea of computing similarity value between two compound labels is inherited from the token-based similarity measures. In fact, a compound label can be split into a set of tokens. Now we can apply a similarity measure, which is a combination of string-based and language-based measures, to compute the similarity score between two tokens. Having the similarity scores between every pair of tokens, we can use the two widely used aggregation methods namely **ExtendedJaccard** and **Monge-Eklan**.

Formally, let `TokenSim`($t_1$, $t_2$) be a similarity measure that compares two tokens $t_1 \in \texttt{tokenize}(s_1)$ and $t_2 \in \texttt{tokenize}(s_2)$; $\theta_{token}$ is a similarity threshold used to determine if two tokens are similar or not. Note that `tokenize`() is a tokenization function. We define several support functions as follows:

$$\begin{aligned}
\texttt{Shared}(s_1, s_2) &= \{(t_i, t_j) | t_i \in \texttt{tokenize}(s_1) \wedge t_j \in \texttt{tokenize}(s_2) : \texttt{TokenSim}(t_i, t_j) \geq \theta_{token}\} \\
\texttt{Unique}(s_1) &= \{t_i | t_i \in \texttt{tokenize}(s_1) \wedge \forall t_j \in \texttt{tokenize}(s_2) : (t_i, t_j) \notin \texttt{Shared}(s_1, s_2)\} \\
\texttt{Unique}(s_2) &= \{t_j | t_j \in \texttt{tokenize}(s_2) \wedge \forall t_i \in \texttt{tokenize}(s_1) : (t_i, t_j) \notin \texttt{Shared}(s_1, s_2)\}
\end{aligned}$$

**Extended Jaccard Formula** An extension of the **Jaccard** similarity is to introduce a weight function `w` for matching and non-matching tokens. For instance, tokens in a pair of `Shared` may get a weight corresponding to their similarity, whereas, tokens in `Unique` get a weight equal to 1.0. Let $\Sigma$ be an aggregation function aggregates the individual weight. The ExtendedJaccard similarity is defined as:

$$ExtendedJaccard(s_1, s_2) = \frac{\Sigma_{(t_i,t_j)\in Shared(s_1,s_2)} w(t_i, t_j)}{\Sigma_{(t_i,t_j)\in Shared(s_1,s_2)} w(t_i, t_j) + \Sigma_{(t_i)\in Unique(s_1)} w(t_i) + \Sigma_{(t_j)\in Unique(s_2)} w(t_j)} \quad (3.1)$$

For example, assume that `TokenSim` is a combination of the **Lin** similarity measure (i.e., an extrinsic-based measure) and **Identical** string-based measure. The similarity score between tokens of two labels $s_1$: "*UM2_DoctoralThesis*" and $s_2$: "*PhdDissertation*" are shown in Table.3.1.

| | UM2 | Doctoral | Thesis |
|---|---|---|---|
| Phd | 0.0 | 0.70 | 0.22 |
| Dissertation | 0.0 | 0.30 | 1.0 |

Table 3.1: Similarity scores between tokens

If select $\theta_{token} > 0.7$ then $Shared(s_1, s_2) = \{(Thesis, Dissertation)\}$. According to formula 3.1, the similarity computed by **ExtendedJaccard** is $\frac{1.0}{1.0+2+1} = 0.25$. However, if $\theta_{token} \leq 0.7$ then $Shared(s_1, s_2) = \{(Thesis, Dissertation), (Doctoral, Phd)\}$. Therefore, the similarity computed by **ExtendedJaccard** is $\frac{1.0+0.7}{1.0+0.7+1+0} = 0.63$.

**Monge - Elkan Formula** The intuition of the **Monge-Elkan** method is that two strings are similar if their tokens are one by one similar. In fact, it matches every token $t_i$ from $s_1$ to the token $t_j$ in $s_2$ that has the maximum similarity to $t_i$ , i.e., where `TokenSim`$(t_i,\ t_j)$is maximal. These maximum similarity scores obtained for every token of $s_1$ are then summed up, and the sum is normalized by the number of tokens in $s_1$. Formally, the Monge-Elkan method is defined as follows:

$$\texttt{MongeElkan}(s_1, s_2) = \frac{1}{|tokenize(s_1)|} \sum_{i=1}^{|tokenize(s_1)|} max\left\{\texttt{TokenSim}(t_i, t_j)\right\}_{j=1}^{|tokenize(s_2)|} \quad (3.2)$$

In the **Monge-Elkan** method, the similarity threshold $\theta_{token}$ is not needed. However, this method is asymmetric. For example. according to formula 3.2, the similarity score between "*UM2_DoctoralThesis*" and "*PhdDissertation*" is $\frac{0.7+1.0+0.0}{3} =$ 0.57; whereas, the similarity score between "*PhdDissertation*" and "*UM2_DoctoralThesis*" is $\frac{0.7+1.0}{2} = 0.85$. To make this method symmetric, we can take the average of the similarity scores such as:

$$\texttt{SymMongeElkan}(s_1, s_2) = \frac{MongeElkan(s_1, s_2) + MongeElkan(s_2, s_1)}{2} \tag{3.3}$$

According to formula 3.3, the similarity score between "*UM2_DoctoralThesis*" and "*PhdDissertation*" is $\frac{0.57+0.85}{2} = 0.71$

### Observations on Hybrid Similarity Measures

Apparently, a hybrid similarity measure is an extension of a token-based similarity measure. Here, the similarity value of two tokens is computed by a combination of a string-based and a language-based similarity measures. Therefore, the hybrid similarity measures can be used to deal with both the type 2 and type 3 of terminological heterogeneity. When two strings have a high number of shared tokens, which are highly similar in syntactic or similar in meaning, the hybrid similarity measure can detect them as matched. But if the number of shared tokens of two strings is small, both token-based and hybrid measure return a low similarity value and they possibly detect these strings as unmatched. Therefore, for types 4 and type 5 of terminological heterogeneity, we need to exploit other feature information of entities in order to discover mappings between them.

## 3.3 Advanced Terminological Similarity Measures

In this section, we present three advanced similarity measures which exploit the integrated features between terminological, structural and instance data to overcome the difficulties of type 4 and type 5 of terminological heterogeneity. In particular, we propose the following measures:

- Information retrieval based similarity measure is an extension of the hybrid method to deal with type 4 of terminological heterogeneity. In the similarity computation, this measure calculates the information content of each token in each label of entities. We will discuss this measure in 3.3.1.

- Context profile based similarity measure use another information retrieval techniques to deal with all types of terminological heterogeneity. In the computation of similarity between entities, this measure compares the textual contexts of the two to-be-matched entities, instead of comparing their labels solely. We discuss this measure in 3.3.2.

- Instance-based similarity measure is based on instances to deal with all types of terminological heterogeneity. We will discuss this measure in 3.3.3.

### 3.3.1 Information Retrieval Based Similarity Measure

The origin of this measure lies in the following observation. In an ontology representing knowledge of a specific domain, some words, which are not "*stop words*" like articles or prepositions, frequently appear with the others in labels of concepts in the domain. For example, in the **conference.owl** ontology in the Conference organization domain, the total number of concepts is 60. Among these concepts, 14 concepts contain the word "*conference*", and 10 concepts contain the word "*contribution*", whereas, other words like "*author*"and "*speaker*" appear only one time. If the word "*conference*" or the word "*contribution*" are found in a compound label, they are highly possible not key words. Instead, they are used to emphasize the specific meaning of the associated words in a specific domain and distinguish the meaning of the associated words in different domain scopes. For example, "*ConferenceDocument*" refers to a type of document which is used in a conference, but is not a type of document used in other domain (e.g. "*OfficeDocument*", "*FinanceDocument*", etc.). Furthermore, in the conference organization domain, when the user talks about a document, they implicitly refer to a conference document. Therefore, word "*Conference*" may be ignored. For example, in the **cmt.owl** ontology, the ontology developers used "*Document*" instead of "*ConferenceDocument*".

The proposed measure was inspired from the comparison methods of documents in information retrieval field. Basically, "*stop words*" firstly are removed from documents. The remaining words are considered as informative words which convey the content of documents. Next, weighted approach is used to assign a weight value to each remaining word. Here, the value for each processing word represents the relative importance of that word in the context of document. Finally, a computation method (e.g., cosine similarity measure) is applied to calculate a similarity score between two documents.

The main difference between labels comparison in ontology matching and 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 should be modified to adapt to label comparison task. In particular, we are going to discuss weight assignment and similarity computation issues which are quite connected to each other.

**Weighting Assignment**

There are many weighted approaches proposed in the information retrieval literature, e.g., word frequency, inverse document frequency, signal weighting, etc [52]. They are mainly based on statistical calculation of the frequency of occurrence of each word in a large corpus. In relation to the ontology matching task, we consider the following issues:

- Firstly, a large number of ontologies describing the same domain is not available. Commonly, only two ontologies in a matching scenario are given. Moreover, because of the high heterogeneity, ontologies may be slightly overlap or may be totally separated. Then, the words used in one ontology may be very different from the words used in the other. Consequently, there may be no benefit to calculate the frequency of words within multiple ontologies.
- Secondly, the weight of a word depends on the ontology that contains that word. As we mention above, common words in a specific domain may explicitly appear many times in one ontology. They also may not appear but be implicitly represented in the other ontologies. Therefore, if we take multiple ontologies in account, the frequency of occurrence of the common words and keywords maybe not strongly different. Consequently, there is not much difference between common words and keywords.

In our approach, a weight value is computed for each word appeared in a given ontology. In particular, we apply the Shannon's information theory [86] in our weighting method. Here, a normalization of the information content of each word is considered as its weight. In Information 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 as follows:

$$IC(t) = \log \frac{|T|}{|N|} \tag{3.4}$$

$$weight(t) = \frac{IC(t)}{\max_{i=1..|T|}\{IC(t_i)\}} \tag{3.5}$$

Where, $|T|$ is a total number of concepts in a given ontology; $|N|$ is a number of concepts whose label contains word $t$.

For example, in the ontology conference.owl, $IC(Conference) = \log \frac{60}{14} = 0.632$; whereas, $IC(Author) = \log \frac{60}{1} = 1.778$ is a maximum information content value.

Therefore, $weight(Conference) = \frac{0.632}{1.778} = 0.355$, $weight(Author) = 1$.

## Similarity Computation

The similarity computation issue concerns the function or method that calculates similarity score between labels. In our approach, an appropriate similarity computation method fulfills two properties as follows:

- **Intelligent**. This property means that the computation method should recognize the amount of informativeness that each token (or word) carries in each label. By knowing the degree of importance of each token, the similarity score between labels relies more on the highly important tokens than on the rest. For example, in the conference organization domain, token "*Participant*" is the most important part in label "*ConferenceParticipant*".
- **Discriminating**. This property means that the computation method rarely assigns the same similarity value when it compares one particular label to several quite similar other labels. For example, it should distinguish the different similarity of "*Publication*" to "*Journal*", "*Magazine*" and "*Periodical*".

According to these requirements, the similarity computation method should take both the weight values of tokens and the similarity values between tokens into account.

**Review of Existing Methods** To the best of our knowledge, ExtendedJaccard and SoftTFIDF are the only two methods that have implemented this idea to compute similarity score between labels. The ExtendedJaccard computation method was shown in Section 3.2.3. Now, we are going to discuss the main features of SoftTFIDF method.

The SoftTFIDF computation method is an extension of cosine similarity based on TFIDF weighted approach. Let $TokenSim(t_1, t_2)$ be an internal terminological similarity measure used to compare tokens; $tokenize(s)$ be a tokenization function. Then, SoftTFIDF defines $Close(\theta_{token}, s_1, s_2)$ is the set of token $t_i$ in $s_1$ such that there is some token $t_j$ in $s_2$ where their similarity score is higher than a predefined threshold $\theta_{token}$. That is:

$$\begin{aligned} Close(\theta_{token}, s_1, s_2) &= \{t_i | ti \in tokenize(s_1) \wedge \exists t_j \in tokenize(s_2) : \\ &\quad : TokenSim(t_i, t_j) > \theta_{token}\} \end{aligned} \tag{3.6}$$

Note that SoftTFIDF is similar to ExtendedJaccard in the way of using a filter threshold $\theta_{token}$ to determine a collection of high similar tokens from two input strings. However, the $Close(\theta_{token}, s_1, s_2)$ in SoftTFIDF only includes tokens from string $s_1$ and not from $s_2$, whereas, the $Shared(s_1, s_2)$ in ExtendedJaccard contains every pair of tokens. Besides, SoftTFIDF is similar to Monge-Eklan method to determine the most similar token $t_j$ in $s_2$ to any token $t_i$ in the $Close(\theta_{token}, s_1, s_2)$. That is:

$$maxsim(t_i, t_j) = \max_{t_j \in tokenize(s_2)} TokenSim(ti, tj) \tag{3.7}$$

Then, the hybrid version of the cosine similarity measure, called SoftTFIDF is defined as:

$$SoftTFIDF(s_1, s_2) = \sum_{ti \in Close(\theta_{token}, s_1, s_2)} \left( \frac{weight(t_i) \cdot weight(t_j) \cdot maxsim(t_i, t_j)}{\|V_1\| \cdot \|V_2\|} \right) \tag{3.8}$$

Here, $V_1$, $V_2$ are the vector representations of $s_1$ and $s_2$. Each numerical value in $V_k$ is a weight of the corresponding token in $s_k$, where $k = 1, 2$. In SoftTFIDF, a weight value is computed by TFIDF (i.e., token frequency-inverse document frequency) weighted approach. In general, the TFIDF approach requires a large corpus of documents to compute weight for each token. Because a large corpus related to a given matching scenario is not always available, we can use our information retrieval based weighted approach instead of TFIDF.

**Discussion of the Existing Methods** Both of the ExtendedJaccard and SoftTFIDF methods have drawbacks. We will discuss their drawbacks through the following examples.

Firstly, ExtendedJaccard lacks of discriminating property. Without loss of generality, assume that weight values of all tokens "*Publication*", "*Journal*", "*Magazine*" and "*Periodical*" are equal to 1.0. Assume that ExtendedJaccard use a language-based measure (e.g., Lin similarity measure) to compare tokens. Table 3.2 shows the similarity scores between them.

| | `Journal` | `Magazine` | `Periodical` |
|---|---|---|---|
| `Publication` | `0.75` | `0.76` | `0.89` |

Table 3.2: Similarity scores between tokens

Obviously, if the filter threshold $\theta_{token} \leq 0.75$, then $Shared(Publication, Journal)$

= {(Publication, Journal)}, $Unique(Publication) = \emptyset$, $Unique(Journal) = \emptyset$ .

$$\begin{aligned} ExtendedJaccard(Publication, Journal) &= \frac{TokenSim(Publication, Journal)}{TokenSim(Publication, Journal) + 0 + 0} \\ &= \frac{0.75}{0.75} = 1.0 \end{aligned}$$

Similarly, we have: $ExtendedJaccard(Publication, Magazine) = 1.0$ and $ExtendedJaccard(Publication, Pediorical) = 1.0$. However, according to their definition in Wordnet, "*journal*" and "*magazine*" are sibling and they both are children of "*periodical*". Therefore, ExtendedJaccard method does not satisfy the discriminating property.

Next, the SoftTFIDF method is asymmetric and maybe returns similarity score higher than 1.0. Assume we compute similarity score between two strings: "*Publication*" and "*PeriodicalPublication*". We have,$Close(Publication, PeriodicalPublication)$ is {(Publication,Publication)}; $Close(PeriodicalPublication, Publication)$ is {(Periodical,Publication),(Publication,Publication)}.

$$\begin{aligned} SoftTFIDF(Publication, PeriodicalPublication) &= \frac{1 \cdot 1 \cdot 1}{1 \cdot \sqrt{1+1}} = \frac{1}{1.41} = 0.70 \\ SoftTFIDF(PeriodicalPublication, Publication) &= \frac{1 \cdot 1 \cdot 1}{1 \cdot \sqrt{1+1}} + \frac{1 \cdot 1 \cdot 0.89}{1 \cdot \sqrt{1+1}} \\ &= \frac{1}{1.41} + \frac{0.89}{1.41} = 1.34 \end{aligned}$$

By this example, we see the similarity score computed by SoftTFIDF depends on the order of input strings. Moreover, if we compute the average similarity score of these values, we obtain $\frac{0.70+1.34}{2} = 1.02$, which is still higher than similarity score between identical strings "*Publication*" vs. "*Publication*". Therefore, this method does not guarantee the reliability of result.

In order to avoid the weaknesses of the existing methods, in the next paragraph, we will present our similarity computation method which is an extension of the Tversky similarity measure [95].

**Extended Tevrsky measure** Here, we propose an extension of Tversky similarity measure to compute similarity score between two strings. First of all, we will illustrate the idea through a simple example to compare "*Publication*" and "*PeriodicalPublication*".

Fig.3.1 shows the relation between two objects A and B. $A \cap B$ is the part shared

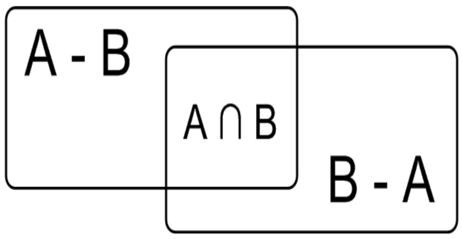


Figure 3.1: A graphical illustration of the relation between features of two objects A and B [95]

between A and B. $A - B$ is the unique part that belong to A only. $B - A$ is the unique part belong to B only. Formally, the Tversky's measure is defined as follows:

$$Tversky_{measure}(A, B) = \frac{f(A \cap B)}{f(A \cap B) + \alpha f(A - B) + \beta f(B - A)} \tag{3.9}$$

When $\alpha = \beta = 0.5$, the Tversky becomes:

$$\begin{aligned} Tversky_{measure}(A, B) &= \frac{f(A \cap B) + f(B \cap A)}{f(A \cap B) + f(A - B) + f(B \cap A) + f(B - A)} \\ &= \frac{f(A \cap B) + f(B \cap A)}{f(A) + f(B)} \end{aligned} \tag{3.10}$$

Now, we define the function $f$ and the shared part, unique parts between token-token as follows:

$$shared(t_i, t_j) = \begin{cases} TokenSim(t_i, t_j) & \text{if } TokenSim(t_i, t_j) \geq \theta_{token} \\ 0 & \text{Otherwise} \end{cases}$$

$$unique((t_i, t_j) = 1 - shared(t_i, t_j)$$

$$f(t_i \cap t_j) = weight(t_i) \cdot shared(t_i, t_j) \tag{3.11}$$

$$f(t_j \cap t_i) = weight(t_j) \cdot shared(t_i, t_j) \tag{3.12}$$

$$f(t_i - t_j) = weight(t_i) \cdot unique(t_i, t_j) \tag{3.13}$$

$$f(t_j - t_i) = weight(t_j) \cdot unique(t_i, t_j) \tag{3.14}$$

We next define the function $f$ for string and for token-string as follows:

$$\begin{aligned} f(s) &= \sum_{t \in s} weight(t) & (3.15) \\ f(t_i \cap s) &= weight(t_i) \cdot \max_{t_j \in tokenize(s)} shared(t_i, t_j) & (3.16) \\ f(s_1 \cap s_2) &= \sum_{t_i \in s_1} f(t_i \cap s_2) & (3.17) \end{aligned}$$

Assume the *TokenSim* be Lin similarity measure, weight value of each token be 1.0 like example in the previous paragraph. The similarity score between "*Publication*" and "*PeriodicalPublication*" is computed as follows:

$$\begin{aligned} f(Publication) &= 1 \\ f(PeriodicalPublication) &= 1 + 1 = 2 \\ f(Publication \cap PeriodicalPublication) &= f(Publication \cap Publication) \\ &= 1 \cdot 1 = 1 \\ f(PeriodicalPublication \cap Publication) &= f(Publication \cap Publication) + \\ &+ f(Periodical \cap Publication) \\ &= 1 \cdot 1 + 1 \cdot 0.89 = 1.89 \end{aligned}$$

Finally,

$$Tversky_{measure}(Publication, PeriodicalPublication) = \frac{1 + 1.89}{1 + 2} = 0.96.$$

### 3.3.2 Context Profile Similarity Measure

Here, we propose context profile measures which are inspired from the idea of finding term-term similarity in the information retrieval field [62]. The intuition is that *"The key to similarity is that two terms appear in the same context - that is they have very similar neighboring terms"*.

To relate to the ontology matching task, we may assume a term is as an entity and neighboring terms are set of other entities that are related to the entity in question. However, the major obstacle here is that term-term similar can be easily and deterministically defined by comparing two strings, but the similarity of entities is unknown or uncertain beforehand. Therefore, leveraging terminological and structural features are needed. Herein, we assume the local name of an entity as a term, its neighboring terms are found as string tokens in labels, comments of itself and other related entities in the ontology. Similar to the original idea in information retrieval field, we call the collections of neighboring terms as the context profile features (or context for short) of an entity.

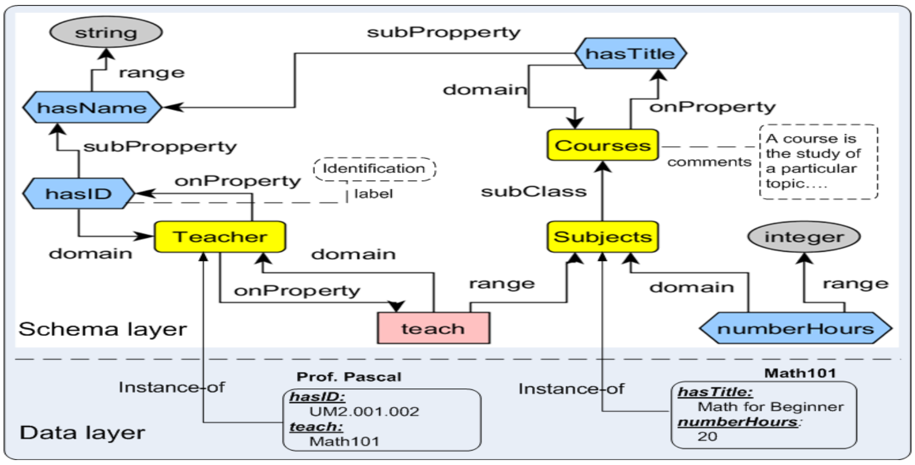


Figure 3.2: Fragment of an ontology

In our system, we construct three different types of context profiles for each entity such as: IndividualProfile, SemanticProfile and ExternalProfile.

**Definition 1.** *IndividualProfile $IP(e)$ of an entity $e$ is a collection of strings included in its name and annotation. Individual Profile $IP(i)$ of an instance $i$ is a collection of strings included in its annotation and property values. The later definition has recursive property, which means that string value of an object property is the individual profile of the corresponding object value.*

**Definition 2.** *SemanticProfile $SP(c)$ of a class $c$ consists of individual profile of itself, its sub-classes and restricted properties. Semantic Profile $SP(p)$ of a property $p$ consists of individual profile of itself, its sub-properties, its domains and ranges.*

**Definition 3.** *ExternalProfile $EP(c)$ of a class $c$ consists of individual profile of all instances belonging to either itself or its descendants. External Profile $EP(p)$ of a property $p$ consists of all data values corresponding to this property appearing in all instances of the ontology.*

From the definition, we have:
$IP(e) = Name(e) \vee Labs(e) \vee Coms(e)$
$IP(i) = \bigcup_{p \in DP(i)} Val(i,p) \vee \bigcup_{p \in OP(i)} IP(Val(i,p))$
$SP(c) = \bigcup_{e \in \{c \cup Desc(c) \cup Rest(c)\}} IP(e)$
$SP(p) = \bigcup_{e \in \{p \cup Desc(p) \cup Doms(p) \cup Rans(p)\}} IP(e)$

$EP(c) = \bigcup_{i \in \bigcup_{e \in \{c \cup Desc(c)\}} Inst(e)} IP(i)$
$EP(p) = \bigcup_{i \in Inst(p)} Val(i, p) \,|\, p \in DP$
$EP(p) = \bigcup_{i \in Inst(p)} IP(Val(i, p)) \,|\, p \in OP$

Where, for every entity $e$, $Name(e)$, $Labs(e)$, $Coms(e)$, $Desc(e)$ are functions returning its local name, labels, comments and descendants respectively; $Doms(p)$, $Rans(p)$ are functions return all domains and ranges of a property $p$; $Rest(c)$ is a function that return all properties, on that a class $c$ has a restriction. $Inst(c)$ is a function returning all instances that belong to class $c$ and $Inst(p)$ is a function returning all instances that have property $p$; $Val(i, p)$ is a function returning a value corresponding to the property $p$ in instance $i$; $OP(i)$ and $DP(i)$ are functions returning a list of object properties and data properties of instance $i$, respectively.

Let us illustrate how to build IndividualProfile, SemanticProfile and External-Profile following a fragment of ontology in Fig.3.2
$IP(Courses)$ = *"Courses A course is the study of a particular topic"*
$IP(hasID)$ = *"has ID identification"*
$IP(Prof.Pascal)$ = *"UM2 001 002 Math for Beginner 20"*
$SP(Courses)$ = *"Courses A course is the study of a particular topic Subject has Title"*
$SP(hasID)$ = *"has ID identification Teacher string"*
$EP(Courses)$ = *"Math for Beginner 20"*
$EP(hasID)$ = *"UM2 001 002"*
$EP(teach)$ = *"Math for Beginner 20"*

Now, the similarity between entities can be computed by the similarity between their context profile features. Similar to work [80], we call each type of context profile above as a virtual document. The similarity between virtual documents is calculated in the Vector Space Model combining with a TFxIDF term-weighting technique. The computation process in detail can be seen in [62]. To sum up, it includes three main phases as follows.

- Phase 1 (term indexing). In this phase, virtual documents of all entities in both ontologies are collected. A refinement process performs tokenization, stop words removing and term stemming for each document. Then, a matrix document-term is built, where each column represents a unique term and, each row represents a document. This matrix is called a vector space model because each of the terms occurring in each document forms a coordinate basis vector of the vector space; thus the dimension denotes the number of all the unique terms and each document is represented as a vector in the vector space.

- Phase 2 (term weighting). In this phase, a term-weighting process assigns weight to each cell in the matrix, which reflects the relatedness between a term and a document. The higher the weight is, the more the term is related to the document. In particular, we apply a similar formula named TFxIDF in [80] for specific term-document such as:

$$weight = TF * IDF$$
$$TF = \frac{w}{W} \quad ; \quad IDF = \frac{1}{2} * \left(1 + \log \frac{N}{n}\right)$$

  Where, $w$ is the frequency of a specific term occurred in a specific document. $W$ is total frequency of all terms in that document. $N$ is a total number of documents and $n$ is a number of documents containing that term.

- Phase 3 (similarity computing). Finally, the similarity between two virtual documents $D_i$, $D_j$ is measured by the cosine value between the two vectors $\vec{V_i}$ and $\vec{V_j}$ representing the two row with index $i$ and $j$ respectively in the vector space model. The measure is as follows:

$$cos(V_i, V_j) = \frac{\sum_{k=1}^{D} n_{ik} n_{jk}}{\sqrt{\sum_{k=1}^{D} n_{ik}^2 \sum_{k=1}^{D} n_{jk}^2}}$$

  where, $|D|$ is the total number of terms contained in the Vector Space Model, $n_{ik}$ and $n_{jk}$ are the coordinate value at $k$th dimension of $V_i$ and $V_j$ respectively.

Let $sim_I(e_i, e_j)$, $sim_S(e_i, e_j)$ and $sim_E(e_i, e_j)$ are functions calculating similarity scores between entities $(e_i, e_j)$ by IndividualProfile, SemanticProfile and External-Profile respectively. To combine all of types of context profiles of entities, we propose the following generic formula:

$$\texttt{sim}(e_i, e_j) = \boldsymbol{f}(sim_I(e_i, e_j), sim_S(e_i, e_j), sim_E(e_i, e_j)) \tag{3.18}$$

Where $\boldsymbol{f}$ may be *weighted average, max, etc.* If the combination function returns only similarity score achieved by SematicProfile, then our context profile similarity measure is similar to measures used in [80, 63, 58]. If the combination function returns only similarity score achieved by ExternalProfile, then our context profile is similar to the instance-based measure.

### 3.3.3 Instance-based Similarity Measure

In this section, we propose an instance-based similarity measure that exploits instance data provided by ontologies to discover mappings between entities. There are two issues we should consider here: (i) how to find similar instances from two ontologies; and (ii) given a list of similar instances, how to discover new concept/property mappings.

For the first issue, we propose two methods for discovering instance mappings as follows:

- If two instances belong to two matched classes and they have highly similar labels then they will be considered as similar. Here, the list of matched classes is taken from the result of Terminological module. Similarity score between instance labels can be computed by a string-based similarity measure described in section 3.2.1. That means:
  `if` $sim(labs(i_1), labs(i_2)) > \theta_1 \,|\, i_1 \in Inst(c_1)$, $i_2 \in Inst(c_2) \,\&\, \langle c_1, c_2 \rangle$ `then` $\langle i_1, i_2 \rangle$

- If two instances have similar description, they will be considered as similar too. Here, a description is taken from values of all properties described in an instance. The intuition behind is that the property values of the same instances will be the same. We can use the similar computing method described in section 3.3.2 to compute the similarity value of two descriptions of two instances. Therefore:
  `if` $sim(IP(i_1), IP(i_2)) > \theta_2$ `then` $\langle i_1, i_2 \rangle$

For the second issue, we apply two methods for discovering concept/property mappings as follows:

- If the text values of two properties of two matched instances are highly similar, then these properties are considered as similar. That means:
  `if` $\langle i_1, i_2 \rangle \,\&\, sim(Val(i_1, p_1), Val(i_2, p_2)) > \theta_3$`then` $\langle p_1, p_2 \rangle$

- If most of their instances of two concepts are matched, then these concepts are matched. That means:
  `if` $sim(Inst(c_1), Inst(c_2)) > \theta_4$ `then` $\langle c_1, c_2 \rangle$

In our system, we don't investigate the matching problem at data layer extensively like some other instance matchers do. Our methods mainly focus to find the changes of ontologies during the versioning and evolution processes. According to [71, 91] the change operations for ontologies can be distinguished between three kinds such as: *Conceptual changes*, *Specification changes* and *Representation changes*. Those operations, for instance, may change the concepts labels, relations, add new properties to a concept or use another ontology representation language, but the content of the data instances is almost maintained. Therefore, we set all $\theta_1 = \theta_2 = \theta_3 = \theta_4 = 0.9$, a high threshold with hope that even entities in the schema layer are changed but the data instances are slightly modified.

## 3.4 Experiments and Evaluations

In this section, we evaluate the performance of our proposed similarity measures through experiments over different data sets used in OAEI campaigns. We design three experiments, which illustrate the following interesting issues: (i) Performance of the Context profile similarity measure, (ii) Performance of the Instance-based similarity measure, and (iii) Performance of the Information retrieval based similarity measure.

### 3.4.1 Performance of the Context Profile Method

The aim of this experiment is to evaluate the performance of the Context profile similarity measure. According to the formula 3.18, different aggregation functions $f$ correspond to different context profile measures. In our approach, we select function $f$, which returns the maximum value of similairy values computed by comparing $IndividualProfile$, $SemanticProfile$ and $ExternalProfile$. That is, two entities will be compared by selecting the most similar context profiles. For example, if two entities have the same descriptions, they are highly similar, despite the fact that their neighbors or their instances are not highly overlapped. In this case, using $IndividualProfile$ is enough. Another example is that if two entities have poor annotation (i.e., only one label) and they do not have any shared instances, their similarity value is only counted by their $SemanticProfile$.

To estimate the benefit of using this measure, we design two matchers as follows. The first matcher consists of only the Identical similarity measure (**ID**), which returns 1.0 if two string are identical, otherwise 0.0. The second matcher consists of the **ID** measure and the context profile similarity measure (**CP**). For the context profile measure, we vary a threshold value from 0.5 to 0.95 to select mappings. In the second matcher, the final matching result is obtained by running a union operator overall the collection of mappings returned by both **ID** and **CP** measures.

In this experiment, we use the Conference data set containing 16 real world ontologies. The reason we select this data set is that we would evaluate the performance of the context profile similarity measure in the real ontology matching case. Table 3.3 shows the matching results obtained by the two matchers overall tests in

| Matchers | Precision | Recall | F-Measure | TP | FP | FN |
|---|---|---|---|---|---|---|
| $ID$ | 0.81 | 0.47 | 0.59 | 143 | 33 | 162 |
| $ID + CP_{0.50}$ | 0.52 | 0.55 | 0.53 | 169 | 156 | 136 |
| $ID + CP_{0.55}$ | 0.57 | 0.55 | 0.56 | 169 | 125 | 136 |
| $ID + CP_{0.60}$ | 0.63 | 0.54 | 0.58 | 166 | 98 | 139 |
| $ID + CP_{0.65}$ | 0.68 | 0.53 | 0.60 | 163 | 78 | 142 |
| $ID + CP_{0.70}$ | 0.71 | 0.53 | 0.61 | 162 | 66 | 143 |
| $ID + CP_{0.75}$ | 0.76 | 0.52 | 0.62 | 158 | 49 | 147 |
| $ID + CP_{0.80}$ | 0.77 | 0.51 | 0.62 | 155 | 42 | 150 |
| $ID + CP_{0.85}$ | 0.79 | 0.49 | 0.61 | 150 | 39 | 155 |
| $ID + CP_{0.90}$ | 0.80 | 0.48 | 0.60 | 148 | 37 | 157 |
| $ID + CP_{0.95}$ | 0.81 | 0.48 | 0.60 | 147 | 35 | 158 |

Table 3.3: Performance of the context profile measure on the Conference data set in OAEI 2009

the Conference data set. Here, $TP$ means the total number of correct discovered mappings, $FP$ means the total number of incorrect mappings and $FN$ means the total number of mappings that matcher did not find. In the first column, $CP_x$ means a threshold value $x$ is used to select mapping result for the context profile measure.

Generally, the second matcher has better *Recall* than the first one. That is, the $CP$ measure can help us to discover new correct mappings that the $ID$ measure cannot. However, the Precision of the second matcher is lower than that of the first one. That is, the $CP$ measure also produces additional incorrect mappings. Therefore, the selection of threshold value is very important. For example, when the threshold value is small (e.g. 0.5), the number of incorrect mappings ($156-33 = 123$) increases faster than the number of correct mappings ($169 - 143 = 26$). That is, using $CP$ measure decreases the overall matching result. On the contrary, when the threshold value is too high (e.g., 0.95), both the number of correct and incorrect mappings slightly increase. In this case, using the $CP$ measure slightly increases the overall matching result ($Fmeasure$ increases $0.60-0.59 = 0.01$). The best threshold is 0.75 or 0.80, where the $Fmeasure$ increase $0.62 - 0.59 = 0.03$.

To sum up, the context profile method can help us to discover new correct mappings. However, the improvement of the overall matching quality strongly depends on the selection of the threshold value for this measure.

### 3.4.2 Performance of Instance-based Similarity Measure

The aim of this experiment is to evaluate the performance of the Instance-based Similarity Measure. As we discussed in Section 3.3.3 above, this measure is designed for discovering mappings of different versions of ontologies. Therefore, in this

| Matchers | Precision | Recall | F-Measure | TP | FP | FN |
|---|---|---|---|---|---|---|
| $ID$ | 0.99 | 0.57 | 0.72 | 4504 | 29 | 3344 |
| $ID + IB$ | 0.98 | 0.62 | 0.76 | 4902 | 97 | 2946 |

Table 3.4: Performance of the instance-based measure on the OAEI 2009 Benchmark dataset

experiment, we select the Benchmark data set published in OAEI 2009. The reason we select this data set is that the test ontology in each test is produced from the original ontology by running a versioning process.

In order to highlight the importance of this measure, we design two matchers as follows. The first matcher consists of only the Identical similarity measure (**ID**). The second matcher consists of the **ID** measure and the Instance-based similarity measure (**IB**). In the second matcher, the final matching result is obtained by running a union operator overall the collection of mappings returned by both **ID** and **IB** measures.

Table 3.4 shows the matching results of the two matchers on the Benchmark data set. Obviously, the $IB$ measure discovers the number of correct mappings ($4902 - 4504 = 398$) more than the number of incorrect mappings ($97 - 29 = 68$). Thus, by using $IB$, the $Fmeasure$ increases ($0.76 - 0.72 = 0.04$). That is, the Instance-based method is useful to discovering mappings of the versioning ontologies.

### 3.4.3 Comparison of Different Similarity Measures for Labels

The aim of this experiment to evaluate the performance of our proposed similarity measure, which is based on information retrieval techniques (**IR**). Because our method focuses on comparing similarity of labels, so in this experiment, we compare the performance of our method with popular methods such as **ISUB**, **Levenstein (Lev)**, **QGrams** that widely used in the existing matching system. Moreover, we also compare it with other methods such as **Token-based Levenstein (TokLev)**, **HybLinISUB** and **HybJCLev**. Here, **TokLev** is a token-based measure which uses Levenshtein measure to compute similarity value of tokens. **HybLinISUB** (or HybJCLev) is hybrid similarity measure that combines **Lin** with **ISUB** measures (or **JiangCorath** and **Levenstein** measures). Their algorithms are described in section 3.2.

In this experiment, the Conference data set in OAEI 2009 is used to evaluate the performance of those measures above. Moreover, a threshold value is used to select a mapping result for each measure. Here, the threshold value varies from 0.6 to 0.95. Fig.3.3 shows the line charts of all similarity measures being compared. Obviously, our similarity measure **IR** outperforms all other measures with respect to all varied

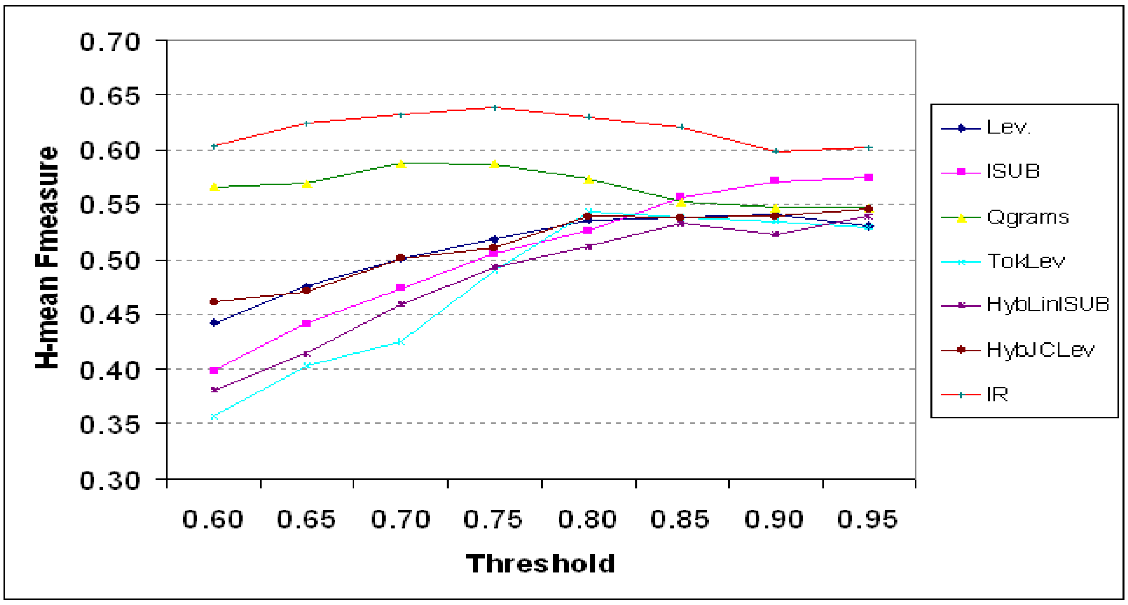


Figure 3.3: Comparison of terminological similarity measures on Conference dataset

threshold values. It obtains the best result when the threshold falls in range from 0.70 to 0.80. This range conforms to what we expected. Assume that there are two entities, one of them has a label containing a single token (e.g., "*AAA*") and the other has a label containing two tokens (e.g. "*AAA BBBB*"). This is a **type 4** of terminological heterogeneity that we discussed in the Section 3.1, and it appears a lot in the Conference data set. Let $x$, $y$ and $z$ are the weights of these three tokens in the two labels. Our IR measure will returns a value $sim = \frac{x+y}{x+y+z} = 1 - \frac{z}{x+y+z}$. According to the heuristic of our IR measure, these two entities are match if the shared token (i.e., "*AAA*" in both labels) are more than the others (i.e., "*BBBB*"). Thus, $\frac{z}{x+y+z} << \frac{1}{3} = 0.33$, consequently, $sim >> 0.66$. That is the range $[0.70, 0.80]$ is expectable.

## 3.5 Conclusion

In this chapter, we have presented a new approach, named IR method to deal with terminological heterogeneity in ontology matching. For this purpose, a comprehensive classification of the different types of terminological heterogeneity has been given. Based on this classification, we have argued that most of the existing terminological similarity measures are able to deal with the first three types (i.e., type 1, type 2 and type 3) of terminological heterogeneity. In particular, they can discover a mapping between two entities if their labels are highly similar in syntax or highly similar in meaning (type 1 and the first part of type 3), or tokens in their labels are pairwise similar in syntax or in meaning (type 2 and the second part of type 3).

However, in real world ontology matching, the last three types appear frequently and are not easy to solve. In these types, labels of the same entities in different ontologies may share only one or few tokens but not all (type 4). Moreover, these labels may be totally different in terms of syntax or in terms of meaning (type 5) or may be represented by different languages (type 6). To overcome these types of heterogeneity, we have proposed three advanced similarity measures, i.e., information retrieval based similarity, context profile similarity and instance-based similarity. Indeed, the Information retrieval based measure has been designed to deal with type 4. To the best of our knowledge, it is the first similarity measure that uses information content of each token to compute the similarity of compound labels of entities in ontologies. This idea lies on the following heuristic: if two entities are the same, the tokens that their labels share are usually keywords of the two labels and have higher information content than the rest. Our experiments show that this measure outperforms all existing similarity measures when dealing with real-case ontology matching.

In addition, a context profile similarity measure has been designed to deal with all types of terminological heterogeneity. In fact, it does not rely on the similarity of the labels of entities, but relies on the similarity of contexts to which entities belong. Our experiments show that by using this measure, we can increase recall. However, the matching quality strongly depends on the selection of a threshold value.

Finally, an instance-based similarity measure has been designed to discover mappings between ontologies in different versions. It makes use of the relationships between concepts, properties and instances to detect new mappings from set of matched instances between two ontologies. Our experiments prove its usefulness in improving the matching quality.

Despite the fact that our similarity measures can adapt to all types of terminological heterogeneity, they cannot discover all mappings completely. It is because they mainly exploit terminological feature encoded in ontology. Using only terminological features is not enough to deal with the two following problems. The first problem relates to the polysemy issue, where two different concepts have the same labels. The second one relates to the synonymy issue, where the same concepts have totally different labels. Generally, the two issues can be solved with the context profile and the instance-based measures. But if the input ontologies do not have instance data, the instance-based measure cannot work. On the other hand, if the annotations of entities in the input ontologies are poor (e.g., ontologies in the Conference data set), the context profiles of entities are poor, too. In that case, the benefit of using Context profile measure is not high. Therefore, to deal with the two problems discussed above, we should use other types of information of an ontology. In particular, the structural information of entities should be exploited. We will discuss this issue in the following chapter.

# Chapter 4

# Dealing with Conceptual Heterogeneity

In the previous chapter, we have argued that using terminological features of entities is not sufficient to discover all mappings between ontologies. A terminology represents a natural visualization of an abstract concept, that evokes its meanings in the real world, but it cannot indicate the exact meaning of this concept in a specific domain of interest. For example, the term "*bank*" has two different meanings with respect to the financial domain and geographical domain. On the other hand, in an ontology, the explicit meaning of a concept is described through a set of semantic relations with other concepts. In addition, its specific semantic meaning is also strengthened by a set of logical statements (i.e., axioms), that indicate the truth of this concept in the domain of interest. For example, the term "*bank*" found in the statement: `Bank is-a Organization`, that supports `Transactions` by `Money`, expresses a concept **Bank** in the financial domain. Because the aim of ontology matching is to discover *semantic mappings* of entities, the explicit meaning is more important and more reliable than the intended meaning of terminology.

Discovering mappings by comparing the semantic description of entities is very difficult because of the high conceptual heterogeneity of ontologies. There are many reasons causing the conceptual heterogeneity. However, they all originate from the different viewpoints of the ontology designers on the domain of interest. Therefore, in section 4.1, we first analyze different types of ontology mismatches caused by the conceptual heterogeneity, and then discuss difficulties to deal with them in the ontology matching task.

Based on the understanding of these challenging issues, in section 4.2, we briefly review the existing structural similarity measures to point out their strengths and weaknesses.

The main contribution of this chapter is found in section 4.3, where a Similarity Propagation method is described in detail. The idea of this method originates from the well-known Similarity Flooding algorithm [67]. Our contribution is to apply this idea to ontology matching field. Moreover, we propose a high level graph data structure for storing semantic relations of entities in an ontology. This data structure helps us to implement the similarity propagation method easily and effectively.

Four experiments in section 4.4 are designed to evaluate different aspects of the similarity propagation method. The experimental results show that this method is stable and improves the overall matching quality.

## 4.1 Analyses of the Conceptual Heterogeneity

Conceptual heterogeneity or semantic heterogeneity of ontologies stands for the mismatches in content of ontologies modeling the same domain. According to [96, 51], ontology mismatches can occur during two sub process in the creation of an ontology such as: *conceptualizing a domain* and *explicating the conceptualization.*

During the conceptualization process, classes, instances, relations, attributes and axioms are distinguished in the domain. Usually, this process also involves ordering the classes and properties in a hierarchical fashion, and assigning attributes and relationships between them. Therefore, a conceptualization mismatch is a difference in the way a domain is interpreted (conceptualized), which results in different ontological concepts or different relations between them.

On the other hand, during the explication process, ontological entities like classes, properties and instances are defined through an ontology language. Each entity is associated a definition, which involves logical formulas or assertion axioms, in order to provide an explicit semantic meaning of the entity in the domain. Therefore, an explication mismatch is different in the way the conceptualization is specified. It can manifest itself in mismatches in definitions, mismatches in terms and combination of both.

According to [96], each conceptualization mismatch is also present in the explication of that conceptualization. In fact, each conceptualization mismatch type occurs in the explication as a definien (type D or CD) mismatch, which means different definitions. However, not all explication mismatches necessarily occur in the conceptualization. To sum up, we survey the main types of ontology mismatches as follows:

- **Class mismatches** are concerned with classes and their subclasses distinguished in the conceptualization.

  - A *categorization mismatch* occurs when the same classes are divided into different subclasses in different conceptualizations. For example, the class `Animal` can be structured around the class `Mammals` and the class `Birds`, but it also can be structured around class `Carnivores` and class `Herbivores`.
  - An *aggregation-level mismatch* occurs when the same classes are defined at different levels of abstraction. For instance, one conceptualization distinguishes class `Persons` and other conceptualization distinguishes class `Males` and `Females` but does not have class `Persons` as their superclass.

- **Relation mismatches** are associated with the relations distinguished in the conceptualization. They concern the relations between classes and the assignment of attributes to classes.
  - A *structure mismatch* occurs when two conceptualizations distinguish the same set of classes but differ in the way these classes are structured by means of relations. For example, in one conceptualization, the class `House` connects to the class `Brick` by the relation `is-made-of`, and in another they are connected by the relation `has-component`.
  - An *attribute-assignment* mismatch occurs when two conceptualization differ in the way they assign an attribute to various classes. For example, two conceptualization distinguish that class `Car` is a subclass of class `Vehicle`. One conceptualization assigns attribute `Color` to `Vehicle`, and the other assigns `Color` to `Car`.
  - An *attribute-type mismatch* occurs when the same attributes are assigned by different instantiations (range of possible values). For example, in one conceptualization, the attribute `Length` assumes its instances to be a number of miles, whereas in another conceptualization, the attribute `Length` assumes its instances to be a number of kilometers.

- **Terminological mismatches** are related to terms (identifiers) to denote concepts in the explication process.
  - A *synonym term* mismatch concerns the same concepts having the same definition, but being represented by different names. For example the term "*car*" in one ontology and the term "*automobile*" in another ontology.
  - A *homonym term* mismatch concerns the different concepts have the same names, but they differ in definitions in different context. For example,

term "*conductor*" has a different meaning in a music domain and in an electric engineering domain.

- **Epistemic mismatches** are associated with different (possibly contradictory) assertions about the same entities in different conceptualizations.

Because of the different types of mismatches discussed above, in reality, even if ontologies describe the same domain knowledge, they are highly conceptually heterogeneous. That makes the process of discovering mappings between entities of real ontologies is very difficult. We can list several reasons for that:

- Firstly, the basic principle of mapping discovery here has a recursive property. Intuitively, two entities are similar if they have been defined by the same definitions, which is containing the same attributes, the same relations to the same other entities. It means that the similarity of two entities depends on and impacts the similarity of other entities related to them and so on.

- Secondly, there does not exist a common structural pattern to recognize similar entitities from different ontologies. Due to high heterogeneity, both the external and internal structure of entities are very different in different ontologies.

  - Here, two entities having the same external structural patterns means the other entities in the same relationship with them are one-by-one similar. In fact, categorization, aggregation and structure mismatches cause the difference of hierarchies and structural relationships between entities in ontologies. Therefore, the possibility of finding the same pattern for two entities is very low. Generally, two entities may be similar on one feature (e.g., same parents) but different on other feature (e.g., different sets of ancestors or different descendants).
  - Similarly, two entities having the same internal structure patterns means their arrtibutes and attribute type of values are similar. However, the attribute-assignment and attribute-type mismatches show that the same concepts may have different attributes and different attribute values. Besides, the implicit inheritance between entities in ontology makes the internal structure of the same concepts different. For example, in the attribute-assignment mismatch, a concept can be directly assigned an attribute, or it can implicitly inherit this property from its ancestor.

- Finally, the automatic discovery methods at conceptual heterogeneity face the uncertainty problem. In fact, a discovering method at structural level is a function of two arguments: initial mappings and structural patterns. Therefore, the uncertainty of a structural method can be caused by the uncertainty of its arguments.
  - Firstly, discovery mappings by exploiting semantic information (i.e., semantic level) of entities requires initial mappings which are usually provided by other matching methods like terminological-based methods. However, because of the terminological mismatches, the matching result of a terminological-based method is not certain. In particular, a homonym mismatch implies that two entities with the same names are different; and a synonym mismatch implies that two entities with different names maybe are the same. Therefore, the initial mappings of the discovery mappings at semantic level are uncertain, consequently, the result of this process might be uncertain either.
  - Secondly, assume that the initial mappings are totally correct, two entities found in the same structural patterns might represent two different concepts. It may be caused by the incompleteness in designing ontologies. For example, in the example of the *categorization mismatch*, class `Mammals`, class `Birds`, class `Carnivores` and class `Herbivores` are subclass of `Animal`. If there is no more detail about them, any mapping between these classes is incorrect.

After doing analyses on conceptual heterogeneity of ontologies, in the next sections, we are going to discuss the similarity methods that can be used to deal with ontology matching at this heterogeneity.

## 4.2 Basic Structural Similarity Methods

According to [28], a basic structural similarity method exploits a specific structural feature of two entities to compare their similarity. This comparison can be subdivided in two types such as:

- Comparison of the internal structure of an entity, which is relating to the *relation mismatches.*
- Comparison of the external structure of an entity, which is relating to the *class mismatches.*

### 4.2.1 Internal Structure Similarity Methods

Internal structure is the definition of entities without reference to other entities. For each ontology class, the internal structure includes its properties (i.e. attributes and relations), range of properties and restricted cardinality on values of properties. For each ontology property, the internal structure includes its domain, range and its own characteristics like transitivity, symmetry, etc.

The internal structure methods are based on internal structure pattern of entities to compute the similarity score between them. Three internal structure patterns as follows are widely used in the ontology matching systems:

- If the properties of two concepts are similar, the concepts are also similar [**?**, 43]
- If the domain and range of two properties are similar, the properties are also similar [**?**, 43]
- If two properties are restricted by the same cardinality (i.e. minimum and maximum cardinality) and have the same type of values, the properties are similar [43].

However, entities with comparable internal structures or properties with similar domains and ranges in two ontologies can be numerous. It makes the internal structure patterns not being strict rule to discover mappings between entities. Therefore, these methods are commonly used to detect the inconsistent mappings rather than to discover accurate correspondences between entities. For example, if two concepts have different cardinality (minimum vs. maximum) on the same properties, they will be considered as two different concepts.

### 4.2.2 External Structure Similarity Methods

External structure (aka. relational structure) is a set of relations that an entity has with its neighbors entities. There are three types of relations that have been considered so far in ontology matching systems such as: (i) *Taxonomic structure* is a backbone hierarchy of an ontology, which is built from `IS-A` relationship between class-class and property-property. For example, **Computer** `IS-A` **Machine**; (ii) *Mereologic structure* is an another type of entities hierarchy, which is built from `PART-OF` relationship. For example, **CPU** `PART-OF` **Computer**; and (iii) *Generic relation structure* is a graph structure, where nodes are classes and edges are properties, which connect classes. For example, **Computer** `connectTo` **Printer**.

Similar to the internal structure methods discussed above, external structure similarity methods compute a similarity score between entities by comparing their external structural features. The intuition behind these methods is that two entities are similar if their neighbors are similar. Several external structure patterns commonly used in ontology matching are as follows:

- ANCESTORS: two entities are similar if all or most of their ancestor entities are similar [19].
- DESCENDANTS: two entities are similar if all or most of their descendant entities are similar [19].
- LEAVES: two entities are similar if all or most of their leaf entities are similar [20].
- ADJACENTS: two entities are similar if all or most of their adjacent entities (parents, children, siblings, domains, ranges) are similar [56].
- ASCOPATH: two entities are similar if all or most of the entities in the paths from the root to the entities in question are similar [56].
- Descendant's Similarity Inheritance (DSIPATH): two entities are similar if the total contribution of the entities in the paths from the root to them is higher than a specific threshold [92].
- Sibling's Similarity Contribution (SSC): two entities are similar if the total contribution of their sibling entities is higher than a specific threshold [92].

Obviously, most of these external structure methods were mainly designed for the *taxonomy structure* of entities in an ontology. Indeed, they can be also used for *mereologic structure* to discover mappings between entities. It is understandable because the external structure patterns discussed above can be found in both hierarchies built on IS-A and PART-OF relations. For example, in the taxonomic structure, if **classX** `IS-A` **classY** and **classY** `IS-A` **classZ**, then **classX** and **classY** are descendants of **classZ**; or **classY** and **classZ** are ancestors of **classX**. Similarly, in the mereologic structure, if **classX** `PART-OF` **classY** and **classY** `PART-OF` **classZ**, then **classX** and **classY** can be considered as descendants of **classZ**; or **classY** and **classZ** can be considered as ancestors of **classX**.

However, these external structure methods cannot apply for the *generic relation structure*. It is because of two reasons. Firstly, generic properties are diverse and they may not be transitive like `IS-A` and `PART-OF` relations. Therefore, in generic structure there does not exist notions of ancestors, descendants or path of entities. Secondly, generic properties are not standard like `IS-A` and `PART-OF` relations. In fact, they are created by ontology designers, so in different ontologies, they may have different representation. Whereas, in different ontologies, `IS-A` and `PART-OF` relations always have the same meaning and the same representation.

### 4.2.3 Discussion on Basic Structure Similarity Methods

There are two advantages of using internal and external structure methods. Firstly, they are simple in similarity computation. Here, both internal and external structure patterns (e.g. ancestors, descendants, path, leaves, etc.) can be easily extracted from ontology. The similarity functions are usually based on operation on set theory (e.g. Jaccard, Dice, etc.). Secondly, they are more reliable than method of discovering mapping by terminology only. It is because they compare the intended meaning of entities in their specific context, which are semantically defined within specific ontologies.

However, they also suffer several disadvantages. At first, they lack discriminating property. For example, many different classes have the same property or have properties with the same datatypes. In this case, internal structure methods cannot distinguish those different classes. On the other hand, when a class has more than one child, some external structure methods based on ancestors or path to root cannot distinguish the different between its children. Therefore, they may return many incorrect mappings. On the other hand, as we mentioned in section 4.1, the similarity between entities computed by structural methods are mutual influence. It means the similarity of two entities are not only depends on its current value but also similarity value of their neighbors. From the beginning, both type of structural methods depend on some initial mappings, which are provided by other matching methods (e.g. terminological methods). If there are some incorrect mappings within the initial mappings, the structural methods may lead to other incorrect mappings.

To overcome the drawback of both types of structure methods, we need to combine their matching evidence to have a more reliable matching result. However, it turns on a well-known challenge in the ontology matching field, namely "*selection and configuration tuning*".

First, selection of a combination function is the first challenge. It is because some structure methods (including internal and external) can work with some entities but cannot work with the others. In fact, as we mentioned in section 4.1, different entities have very different internal and external structure. Therefore, it is not easy to find an appropriate function that takes all structure methods above as its arguments.

Second, setting configuration or parameter for the combination method is another challenge. Here, for each entity, different methods exploit different structure features, which bring different degree of importance in similarity computation. The degree of importance of the same feature ( ancestor, path, etc.) are even very different between different entities. Therefore, manually setting parameter seems to be not applicable.

Furthermore, non-iterative combination method is still error prone. In fact, if initial mappings are incorrect, the structure methods will produce other incorrect mappings, and so on for their combination. Both individual structure methods and combination method do not remove the incorrect mappings from the initial mappings.

For that reasons, in the next section we will present our method, which is an extension of similarity flooding algorithm [67] used in schema matching. In the proposed method, we can take advantage of almost all types of structural feature (i.e., internal and external) in commutation of similarity. Moreover, through the iterative process, the error prone problem can be solved.

## 4.3 Similarity Propagation Method

In this section, we present a similarity propagation method for discovering mappings between entities of two to-be-matched ontologies. Before going in detail about its algorithm, we firstly clarify what is similarity propagation.

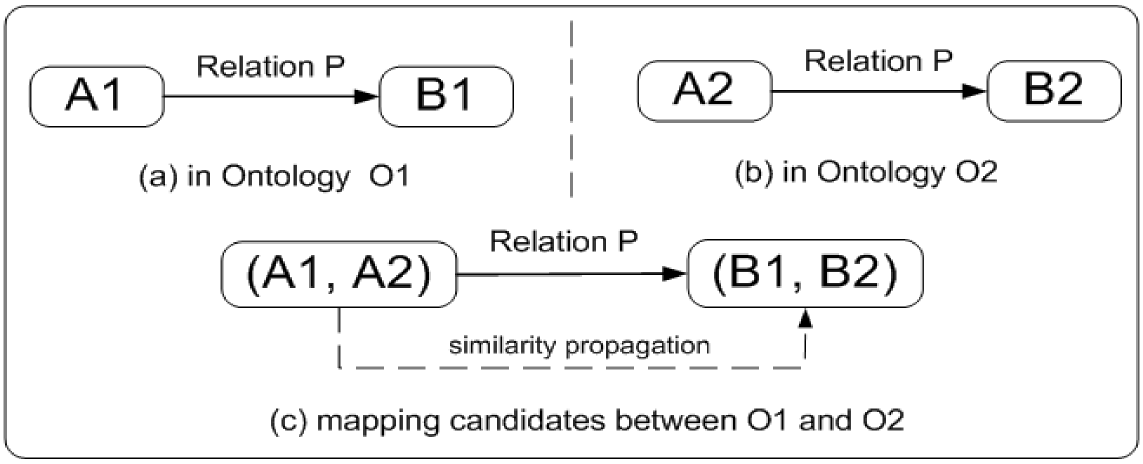


Figure 4.1: Similarity propagation

Let us see Fig.4.1, assume that in the ontology $O1$, two entities $A1$ and $B1$ are connected by a directed relation $P$ (Fig.4.1a). Similarly, in the ontology $O2$, two entities $A2$ and $B2$ are also connected by same relation $P$ (Fig.4.1b). Here, the relation $P$ may be any semantic relationship like `IS-A`, `PART-OF`, `domain`, `range`, or user specific property, etc.

According to the direction of the relation $P$, those entities produce two candidate mappings, i.e., ($A1$, $A2$) and ($B1$, $B2$), which are connected by the relation $P$ (Fig.4.1c). We may say that two candidate mappings are adjacent through the relation $P$. Intuitively, if one of the candidate mappings is correct match, the other mapping is very likely to be a match and vise versa. It means that the similarity of one mapping influences the similarity of the other. Depending on the function of similarity computation, their similarity values will be fully or partly propagated to each other in order to recompute the new similarity score values.

Obviously, the key idea behind the similarity propagation method is similar to the other structure similarity methods, in terms of two entities of two distinct ontologies are similar if they relate to the other similar entities. However, there are three main differences between them as follows.

In contrast to basic structure similarity methods, the similarity propagation method uses a fix point computation, in which the similarity scores are computed iteratively until the global condition point is reached. In the similarity propagation process, the new similarity score of two entities depends not only on the current similarity score between them, but also also on the current similarity scores of their adjacent. Obviously, basic structure similarity methods can be assumed as a similarity propagation method with only one iteration.

Another difference is about the amount of similarity values to be propagated during the similarity computation of two entities from their adjacent. In basic methods, the similarity values will be fully propagated from the adjacent entities. Except, in DSIPATH and SSC methods, the authors use a factor value to distinguish different types of adjacent (e.g., the contribution of parent is more important than the contribution of grandparent and so on). In the similarity propagation method, the similarity value is only partly and directly propagated from two entities to direct adjacent through their relations.

Finally, the similarity propagation method is able to exploit all structure information of entities in an ontology, whereas, basic methods exploit only a part of it. In particular, internal structure methods are limited by class-properties relations (e.g. hasProperty, domain, range); external structure methods are limited by class-class relations (e.g., IS-A, PART-OF). In similarity propagation method, any type of relation between entities can be used to couple two entities of two different ontologies to a candidate mapping. For example, it can couple two classes and two properties to two candidate mappings, which are connected by a class-property relation.

Moreover, it can also couple two pairs of classes from two ontologies to two candidate mappings, which are connected by a class-class relation. Therefore, in [28], the similarity propagation is referred as global based method, whereas, basic structure methods are referred as local based methods.

Now we are going in detail into the similarity propagation method. The main steps in this method are shown in the Algorithm 2. First, input ontologies are transformed into directed labeled graphs *OntoGraph*, then they are merged into a pairwise connectivity graph ($PCG$). Here, the amount of similairy to be propagated is defined by the current similarity score hold in `PCG`'s nodes and the weight values on the edges. Therefore, at the beginning, edges in the `PCG` are assigned weight values by the `Weighted` function and nodes in `PCG` are assigned similarity values taken from initial mappings $M_0$. After initiating values, the `PCG` becomes an induced propagation graph `IPG`. During the `Propagation` on `IPG`, only similarity score hold on nodes are changed, whereas, the edges' weights are not. At the end of each iteration in `Propagation`, all 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 : $O_1$ , $O_2$ : ontologies to be matched
        $M_0 = \{(e_1, e_2, \equiv, w_0)\}$ : initial mappings
output: M $= \{(e_1, e_2, \equiv, w)\}$ : result mappings
1 $G_1 \leftarrow$ Transform($O_1$);
2 $G_2 \leftarrow$ Transform($O_2$);
3 $PCG \leftarrow$ Merge($G_1, G_2$);
4 $IPG \leftarrow$ Initiate($PCG$, Weighted, $M_0$);
5 Propagation($IPG$, Normalized);
6 $M \leftarrow$ Filter($IPG, \theta_s$);
```

In particular, the following issues will be considered, i.e., (i) computation space (i.e., lines number 1,2 and 3), (ii) iteratively computing similarity with propagation (i.e., lines number 4 and 5), and (iii) filters (line 6).

### 4.3.1 Computation Space

A computation space here means the total number of candidate mappings involved in the similarity propagation process. In Fig.4.1, apparently, a similarity value is only propagated from one candidate mapping to another candidate mapping. Therefore, the aim of building a computation space is to determine a collection of candidate mappings and the relationships between them. This step is corresponding to the step of building pairwise connectivity graph (PCG) in the original similarity flooding algorithm. In particular, each node in the pairwise connectivity graph is a candidate mapping produced from two entities of two ontologies.

Basically, two entities of two distinct ontologies become a candidate mapping if and only if they have a same relation to two other entities in those ontologies. The rule to build a pairwise connectivity graph from two distinct ontologies is as follows:

$$((x, y), p, (x', y')) \in PCG(O_1, O_2) \Leftrightarrow (x, p, x') \in O_1 \;\wedge\; (y, p, y') \in O_2$$

where, $x$, $x'$ are two entities of ontology $O1$; $y$, $y'$ are two entities of ontology $O2$; $(x, p, x')$ means entity $x$ heads to entity $x'$ by relation named $p$.

Therefore, in order to avoid the loss of candidate mappings, we need to transform structural information of entities (including internal and external) of two ontologies into the same set of predefined relations. Hence, each ontology is converted to a directed acyclic graph, whose nodes are ontologies entities (i.e., classes, properties and datatypes) and each directed edge between nodes has a label encoded for semantic relationships between the corresponding entities.

Naturally, an ontology can be seen as a RDF graph[1], whose each edge is a RDF triple. That means, for each edge in the graph, `Subject` and `Object` are two end nodes, the direction is heading from `Subject` to `Object` with label `Predicate`. Let us see a fragment example of an ontology and its corresponding RDF graph in Fig. 4.2. Here, **Author** and **Reference** are classes; **hasAuthor** is an object property, whose range is class **Author**; and **hasTitle** is a data property, whose range is a **string** datatype. Class **Reference** has restriction on both **hasAuthor** and **hasTitle**.

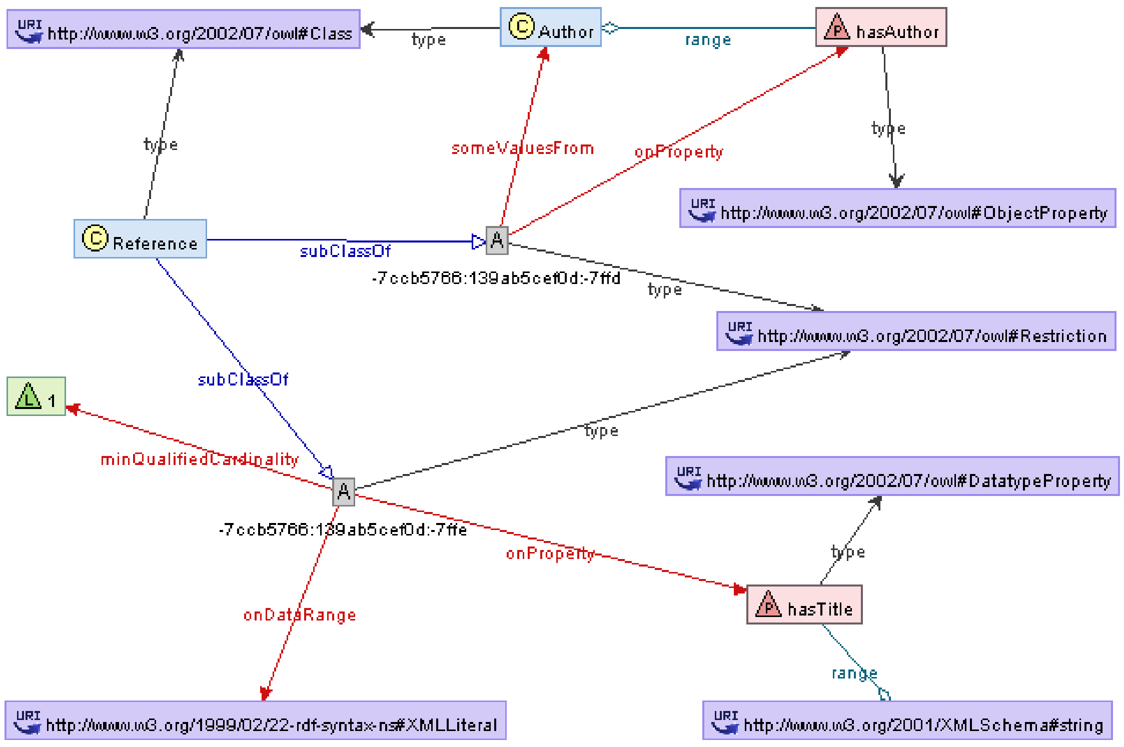
However, this representation obviously has disadvantages such as:

- Two RDF graphs may produce a huge number of redundant nodes in the *PCG*. For example, with the same label `rdf:type`, *PCG* may contain many nodes compounded of classes of the first ontology with properties of the second ontology and vice versa. These nodes are not needed because we try to discover mappings between class-class and property-property only.

- It would produce many incorrect mapping candidates. For example, two RDF triple statements describing classes `Author` and `Reader` in two distinct ontologies as follows:

$$\langle \texttt{Author}, \texttt{rdf:type}, \texttt{rdfs:Class} \rangle \quad \textit{and} \quad \langle \texttt{Reader}, \texttt{rdf:type}, \texttt{rdfs:Class} \rangle$$

[1]http://www.w3.org/TR/rdf-concepts/



http://www.w3.org/2002/07/owl#Class
type
Author
range
hasAuthor
type
someValuesFrom
onProperty
type
http://www.w3.org/2002/07/owl#ObjectProperty
Reference
subClassOf
A
-7ccb5766:139ab5cef0d:-7ffd
type
http://www.w3.org/2002/07/owl#Restriction
subClassOf
1
type
minQualifiedCardinality
http://www.w3.org/2002/07/owl#DatatypeProperty
A
-7ccb5766:139ab5cef0d:-7ffe
onProperty
type
hasTitle
onDataRange
range
http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral
http://www.w3.org/2001/XMLSchema#string

Figure 4.2: A RDF graph for a fragment of an ontology

Because these triples have the same predicates and objects, so it may infer that classes `Author` and `Reader` are similar, which is incorrect.

- Lastly, the RDF graph faces a problem with anonymous (blank) nodes (e.g., the gray square with label A in Fig.4.2 are anonymous nodes). In RDF graph, anonymous nodes are used to describe a complex description of a concept. Unfortunately, we cannot compute the similarity between blank nodes. If we remove them, we loose the connection between nodes using these blank nodes as bridge.

To overcome the weaknesses of the RDF graph, strong constraint conditions on merging edges are needed. For example, in [98], the author proposed a "*Strong Constraint Condition for Similarity Propagation in Triples*" as follows: Given two triples $t_i = \langle s_i, p_i, o_i \rangle$ and $t_j = \langle s_j, p_j, o_j \rangle$, and let $S_s$, $S_p$ and $S_o$ denote the corresponding similarities of $(s_i,\, s_j)$, $(p_i,\, p_j)$ and $(o_i,\, o_j)$ for the two triples. The similarity can be propagated if and only if $t_i$ and $t_j$ satisfy the following three conditions:

- If $t_i$ includes ontology language primitives, the corresponding positions of $t_j$ must be the same primitives

- $t_i$ or $t_j$ has at most one ontology language primitive

- In $S_s$, $S_p$ and $S_o$, at least two similarities must be large than threshold $\theta$ (equal to 0.005 by default)

The first condition ensures that the propagation is only performed when two triples use the same ontology language primitive to describe the facts. Here, the ontology language primitives refer to RDF vocabularies and OWL vocabularies. Therefore, there will be not candidate mappings created by class-property or property-class entities. The second condition ensures that there is no definition and declaration of triples during propagating, because such triples may cause incorrect matching results. The third condition is a heuristic to reduce the computational space by limiting number of candidate mappings with a similarity threshold $\theta$.

However, because these constraints are applied on RDF triples, the problem of anonymous nodes still remains. Generally, if a class in an ontology has a complex description, it is a subclass of some anonymous nodes. For example, in Fig.4.2, class **Reference** is described by as subject of two statements, whose predicate is **subClassOf** and objects are two anonymous nodes namely `-3f45da6:139a10f5a3a:7ffd` and `-3f45da6:139a10f5a3a:7ffe`. Two statements like that will merge not only named ontology entities (i.e. classes, properties) but also their anonymous nodes. Therefore, they will produce a lot of computation nodes in the computation space.

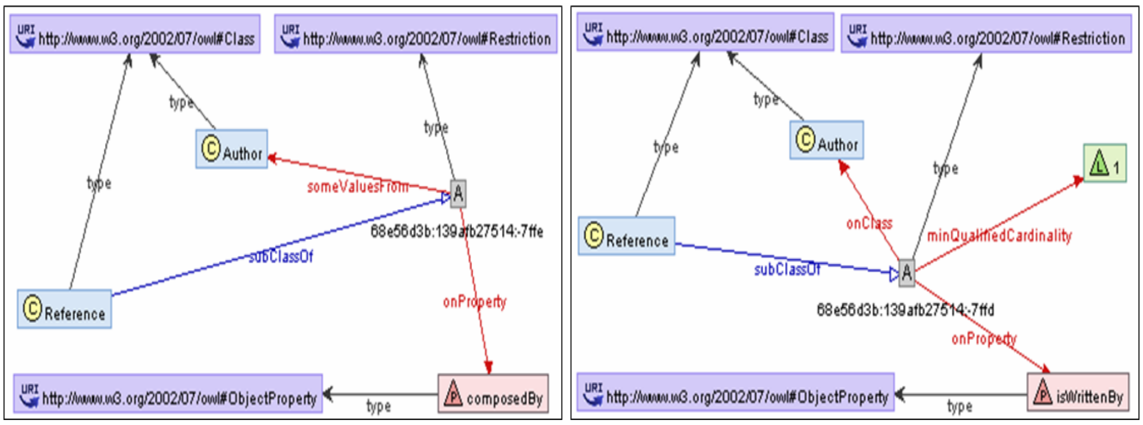

http://www.w3.org/2002/07/owl#Class
http://www.w3.org/2002/07/owl#Restriction
type
Author
someValuesFrom
A
68e56d3b:139afb27514:-7ffe
subClassOf
Reference
onProperty
http://www.w3.org/2002/07/owl#ObjectProperty
composedBy
onClass
minQualifiedCardinality
1
68e56d3b:139afb27514:-7ffd
isWrittenBy

Figure 4.3: Two RDF graphs for two fragments of ontologies

Moreover, computing a similarity score between anonymous nodes is not an easy task. In fact, similarity of anonymous nodes depends on the statement, which the anonymous is involved. Let us see Fig. 4.3, two classes **Reference** in two ontologies are described by two different anonymous nodes namely `68e56d3b:139afb27514:7ffd` and `68e56d3b:139afb27514:7ffe`. These two anonymous nodes have the connection to class **Author** but with different edges (`onClass` vs. `someValueFrom`); or have the same edge (`onProperty`) but connect to properties with different labels

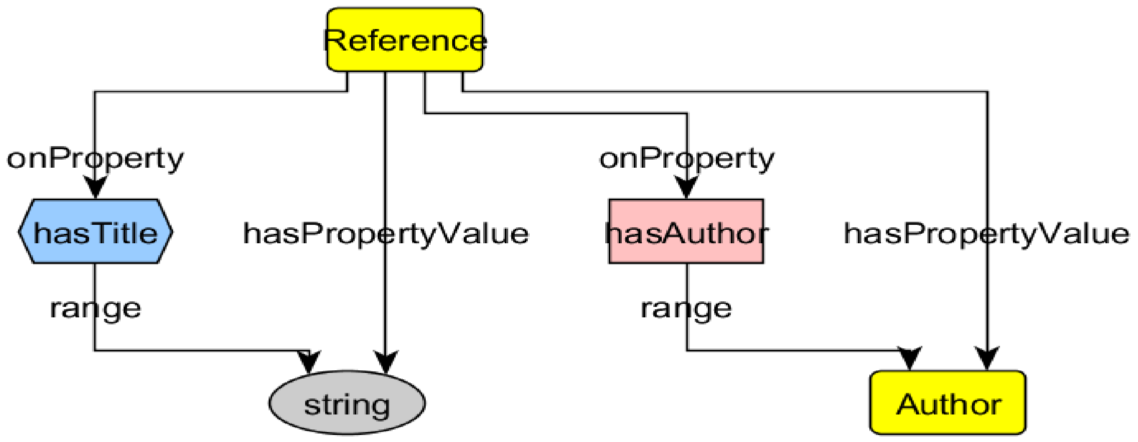


Figure 4.4: A high level ontology graph

(**isWrittenBy** vs **composedBy**). These differences may make two anonymous nodes be very different, consequently make two classes **Reference** be different also.

According to the analyses discussed about RDF graph above, we may conclude that this type of graph is not suitable for the similarity propagation algorithm. To overcome the problem related to anonymous nodes, we extract only main information of each ontology entity to build a directed acyclic graph with predefined labels on edges. In particular,the following rules for building a high level ontology graph will be used:

- For each class, its direct connected classes through IS-A and PART-OF relations are extracted to build a high level ontology graph, where edges between them are assigned with labels `subClass` and `partOf` respectively. Note that, the PART-OF relation is usually used in anatomy ontology in the form `SubClassOf(clsA ObjectSomeValuesFrom(PART-OF clsB))` .Additionally, its properties (i.e., data property, object properties) and their value types are also extracted. In graph, edges between them are assigned with labels `onProperty` and `hasPropertyValue` respectively.

- For each property, its direct parent and children are extracted. Im graph, edges between them are assigned label `subProperty`. Besides, its inverse and equivalent properties are also extracted. In graph, edges between them are assigned labels `inverse` and `equivalent` respectively. Additionally, its domain and range are also extracted. In graph, edges between them are assigned labels `domain` and `range` respectively.

Based on these rules, the fragment of ontology described in Fig.4.2 can be converted to a high level ontology graph as in Fig.4.4. We call the graph a high level graph because we encode the semantic meaning of relations by human understandable. Now, each edge in a high level ontology graph (ontology graph for short) has format:

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

where, `sourceNode` and `targetNode` are ontology entities (i.e., concepts, object properties, data properties) or primitive datatypes. The semantic meaning of an edge is expressed by `edgeLabel`, which belongs to one of the 5 following types: `subClass`, `partOf` `onProperty`, `hasPropertyValue`, `subProperty`, `inverse`, `equivalent`, `domain`, and `range`.

The transformation from ontology to a high level ontology graph brings the following advantages. Firstly, we can easily merge two ontology graphs into a pairwise connectivity graph, which is a computation space for the similarity propagation method. It is because in both graphs, their edges were standardized with predefined labels. Therefore, a pairwise connectivity graph can be retrieved by merging the corresponding source and target nodes of edges with the same label. A node in the pairwise connectivity graph is a candidate mapping created from two entities of two input ontologies. Next, by using only main relations of entities, the computation space will be reduced. In fact, the other internal structure information will be used in the refinement process to detect and reject incorrect mappings. Lastly, it is easy to add new edges, which are not shown explicitly as semantic relations between entities in ontology but can be inferred by a description logic reasoner (e.g., Pellet, Hermit). For example, a class inherits all properties of its super class. Then we only have to add new edges with labels `onProperty` from this class to the properties of its super class. This advantage can be used to deal with the *attribute mismatch* and *class mismatches* as we mentioned in section 4.1

For illustration purpose, the high level graphs of the source and the target ontologies, which were introduced in Chapter 1, are depicted in Fig. 4.5 and Fig. 4.6, respectively. Next, by merging the two ontology graph, we obtain a pairwise connectivity graph (PCG). Fig. 4.7 shows a fragment of the PCG around a candidate mapping (**Staff**, **Employee**). In the next section, we will present how similarity values are propagated in the pairwise connectivity graph.

### 4.3.2 Computing Similarity with Propagation

The process of similarity propagation in our method is inherited from the original similarity flooding algorithm. In the similarity computation process, two issues

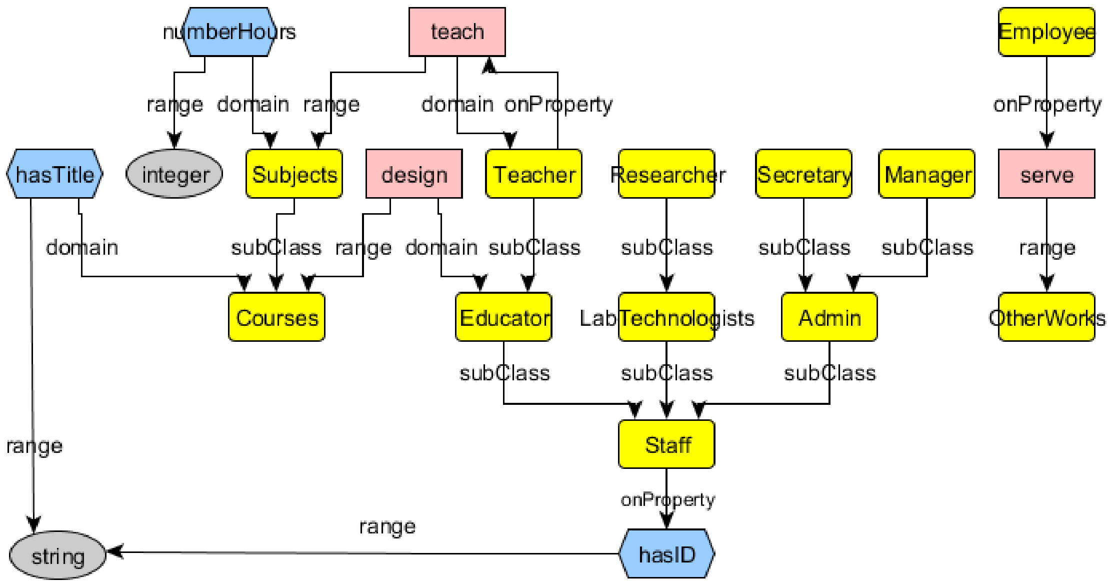


Figure 4.5: Relations between concepts, object properties, data properties in the source ontology

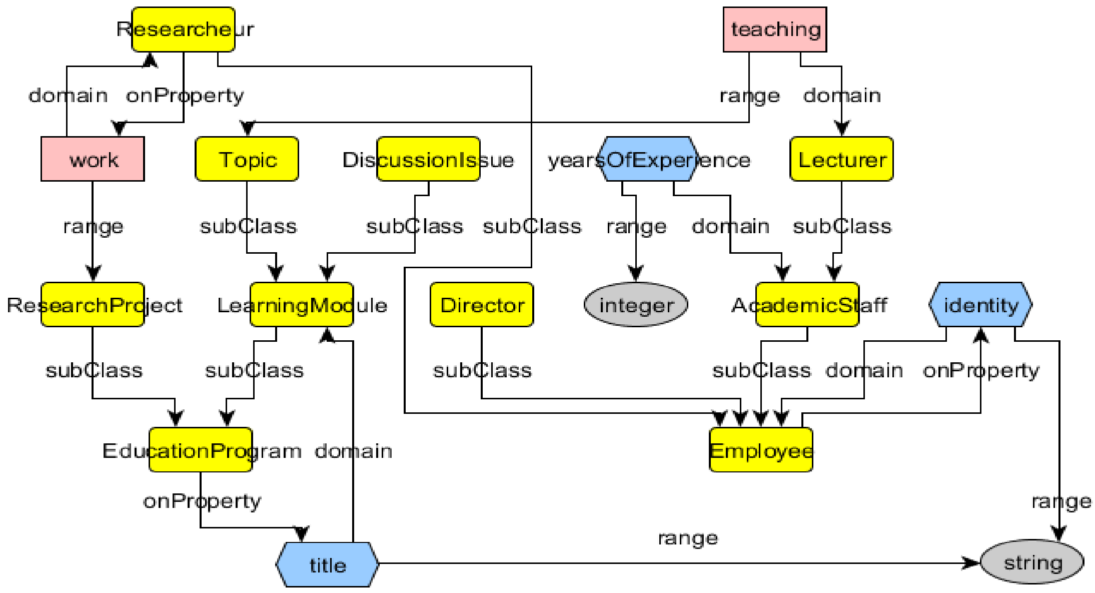


Figure 4.6: Relations between concepts, object properties, data properties in the target ontology

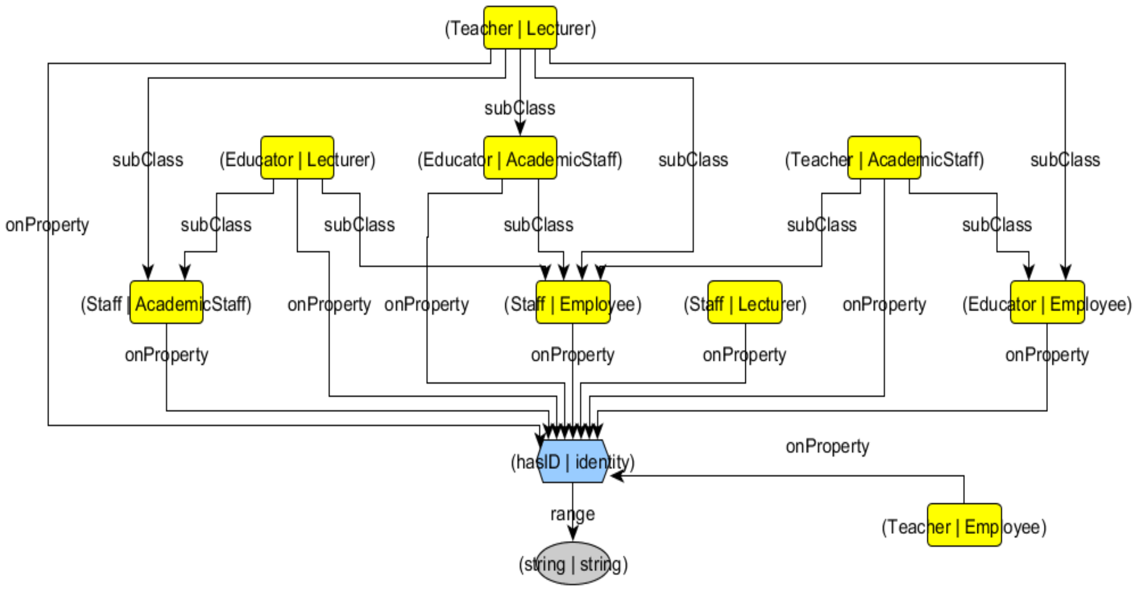


Figure 4.7: A fragment of the pairwise connectivity graph

will be considered: (i) what amount of similarity is propagated from one candidate mapping to the other; and (ii) how to update the similarity score of candidate mappings at each iteration of similarity computation. The first issue is related to the setting weight to every edges in the pairwise connectivity graph. The second issue is related to the updated (i.e., accumulated and normalized) similarity functions used in the propagation process.

**Edge weighting in pairwise connectivity graph** As in the original similarity flooding algorithm, amount of similarity propagated from one node to another in the pairwise connectivity graph depends on the importance of the edge connected between them. In the similarity propagation, the importance of an edge is assigned by a value called propagation coefficient. Intuitively, the propagation coefficient of an edge is determined by two features: the semantic meaning of the relation itself (e.g., `subClass` relation is somehow more important than `onProperty` or `hasPropertyValue` relations), and the frequency of the type of edge adhered to a graph node.

For the first feature, we can set a $FACTOR$ value for each type of semantic relation. For the second feature, we can use one of the computing propagation approaches described in the original algorithm.

In particular, assume $card(x, p, G)$ delivers the numbers of edges of node $x$ that carry label $p$ in graph $G$. The number of outgoing and incoming edges are calculated as follows:

$$card_i(x,p,G) = \|\{(x,p,t)\}\|\exists t : (x,p,t) \in G\|$$

$$card_o(x,p,G) = \|\{(t,p,x)\}\|\exists t : (t,p,x) \in G\|$$

Assume, function $\pi$ defines the propagation coefficients for a candidate mapping $(x, y)$ in the pairwise connectivity graph, where $x$, $y$ are nodes in the high level ontology graphs $G_1$, $G_2$ respectively. Some possibility for a choice of function $\pi$ can be seen in Table 4.1.

| $\pi_{\{i,o\}}((x,p,G_1),(y,q,G_2))$ | $p = q$ |
|---|---|
| InverseAverage | $\frac{2}{card_{\{i,o\}}(x,p,G_1)+card_{\{i,o\}}(y,q,G_2)}$ |
| InverseProduct | $\frac{1}{card_{\{i,o\}}(x,p,G_1)\cdot card_{\{i,o\}}(y,q,G_2)}$ |
| InverseTotalAverage | $\frac{2}{card_{\{i,o\}}(p,G_1)+card_{\{i,o\}}(q,G_2)}$ |
| InverseTotalProduct | $\frac{1}{card_{\{i,o\}}(p,G_1)\cdot card_{\{i,o\}}(q,G_2)}$ |
| CombinedInverseAverage | $\frac{4}{(card_{\{i,o\}}(x,p,G_1)+card_{\{i,o\}}(y,q,G_2))\cdot(card_{\{i,o\}}(p,G_1)+card_{\{i,o\}}(q,G_2))}$ |
| Stochastic | $\frac{1}{\sum_{\forall p'}(card_{\{i,o\}}(x,p',G_1)\cdot card_{\{i,o\}}(y,p',G_2))}$ |
| Constant | 1.0 |

Table 4.1: Different approaches to computing propagation coefficients

Note that, when $p \neq q$, the propagation coefficient function $\pi$ alway returns 0. According to the studies in [67, 98, 31], approaches based on inverse average and inverse product are commonly used and slightly better than other approaches.

Let us see an example of computing propagation coefficient of edges by inverse product approach in Fig.4.7. There are two outgoing `subClass` edges from node (**Teacher**, **Lecturer**). Thus, the propagation coefficient for each of those edges is equal to $\frac{1}{2} = 0.5$. It means that node (**Staff**, **Employee**) receives only 0.5 of similarity values propagated from node (**Teacher**, **Lecturer**).

On the other hand, node (**Teacher**, **Lecturer**) is also influenced by node (**Staff**, **Employee**). However, there are 4 incoming `subClass` edges to node (**Staff**, **Employee**). Thus, the weight or propagation coefficient for each of those edges is equal to $\frac{1}{4} = 0.25$. It means that node (**Staff**, **Employee**) propagates only 0.25 of its similarity values to its adjacent (i.e., (**Educator**, **AcademicStaff**), (**Educator**, **Lecturer**), (**Teacher**, **AcademicStaff**) and (**Teacher**, **Lecturer**)) through `subClass` edges.

Therefore, the `subClass` edge from node (**Teacher**, **Lecturer**) to node (**Staff**, **Employee**) has outgoing coefficient of 0.5 and has incoming coefficient of 0.25.

| Identifier | Updated similarity function |
|---|---|
| **Basic** | $\sigma^{(i+1)} = normalize(\sigma^i + \varphi(\sigma^i))$ |
| **A** | $\sigma^{(i+1)} = normalize(\sigma^0 + \varphi(\sigma^i))$ |
| **B** | $\sigma^{(i+1)} = normalize(\varphi(\sigma^0 + \sigma^i))$ |
| **C** | $\sigma^{(i+1)} = normalize(\sigma^0 + \sigma^i + \varphi(\sigma^0 + \sigma^i))$ |

Table 4.2: Variation of the updated similarity functions

**Updated similarity function** In each iteration of the similarity propagation process, every node in graph (i.e., a candidate mapping) will update its similarity score. As described in the original algorithm, the updated similarity function for each node is performed in two steps: (i) accumulation similarity propagated from its adjacent; and (ii) normalization similarity values over the entire computation space.

Firstly, the accumulation of propagated similarity values is performed as follows. Assume $\sigma^i(a, b)$ is the similarity score value of two entities $a \in G_1$ and $b \in G_2$ at iteration $i$. In the next iteration $(i + 1)$, the generalized function to accumulate similarity score values is:

$$\begin{aligned}\varphi(\sigma^i(a,b)) &= \sum_{(a,p,x)\in G_1,(b,q,y)\in G_2} \sigma^i(x,y)\cdot\pi_o((x,p,G_1),(y,q,G_2)) + \\ &+ \sum_{(x,p,a)\in G_1,(y,q,b)\in G_2} \sigma^i(x,y)\cdot\pi_i((x,p,G_1),(y,q,G_2))\end{aligned}$$

Secondly, the normalization function projects similarity values of all candidate mappings into the range [0,1]. According to the original algorithm, a normalized similarity value is obtained by dividing the current value by the highest similarity value in the computation space. In particular:

$$normalize(\sigma(a,b)) = \frac{\sigma(a,b)}{max\,\{s\|\exists x \in G_1, y \in G_2 : \sigma(x,y) = s\}}$$

Finally, the updated function, which is a combination of accumulation and normalization functions, can be defined as one of the formulas in Table 4.2.

The updated similarity function is known as a fixpoint function in each iteration of the similarity propagation. It directly impacts the convergence and efficiency of this process. According to the study in the original algorithm [67], the formula **C** makes the propagation converge fast and it does not negatively impact the quality of the results.

### 4.3.3 Filters

Mapping filter is to choose the best match candidate from the list of possible mappings. Several filters such as threshold filter, greedy filter or maximum weighted assignment filter can be used after performing similarity propagation. We will discuss them all in detail in the section 6.1 in Chapter 6. Here, in context of the similarity propagation method, we apply two consecutive filters, i.e., threshold filter then maximum weighted assignment filter. The first one eliminates low similarity candidate mappings, whereas the second one guarantee the best solution for 1:1 matching cardinality.

## 4.4 Experiments and Evaluations

In this section, we evaluate the different aspects of our similarity propagation method. In particular, we first compare the performance of this method against the other structural methods. Furthermore, we investigate the impact of input noise on this method and the other structural methods. Next, the impact of quality of input on this method is also studied. Finally, we evaluate the impact of using a reasoning system to detect hidden relationships between entities in ontologies on the performance of the similarity propagation method.

### 4.4.1 Comparison of the Similarity Propagation Method with Basic Structural Methods

In this evaluation, we are going to compare the effectiveness of using our Similarity Propagation method (SP) to other existing structural methods. In particular, all the basic external structural methods described in section 4.2 will be used in the comparison, i.e., ANCESTORS, DESCENDANTS, LEAVES, ADJACENTS, ASCOPATH, DSIPATH and SSC.

To perform this experiment, we have used Benchmark 2011 dataset including 103 test cases. These test cases are mainly considered for structural evaluation because of the following features: (i) Because entities do not have annotation (i.e., labels, comments) and their names are altered by random strings (no variation by naming convention or synonym words), the combination of different string based, linguistic based methods are not necessary. In this experiment, we can use only a simple string method to check whether two strings are identical or not. The interesting point here is that if two entities from two input ontologies have the same name, they are a correct mapping; (ii) In some tests, the structure of ontologies are not changed but a number of names are replaced by random strings. In other tests, not only names of entities are altered but also the ontology structure is changed (flatten, extension, etc.).

According to this observation, the matching strategy used in this experiment is described as follows:

- Only 3 modules will be used: Element based matcher, Structure based matcher and Mapping selection.
- Element based matcher provides init mappings to structural based matcher. It is based on the Identical similarity measure (see section 3.2.1 in Chapter 3).
- Each structure based matcher corresponding to each of the structural methods selected above produces a similarity matrix for all pairs of entities of the two input ontologies.
- We vary different threshold (0.01 - 0.9) to select mappings discovered by structural matcher. The mappings obtained by structural matcher are combined with the mappings obtained by element based matcher to produce the set of candidate mappings. Then, a greedy selection method described in section 6.1 in Chapter 6 is used to extract the final alignment.

Obviously, when the threshold varies from 0.6 to 0.9, the structural method lines in Fig. 4.8 seem to be converging into INIT-MAPPINGS line where H-mean Fmeasure = 0.68 (4643 correct mappings, 27 incorrect mappings, 4342 unfound). It means that the structural methods did not discover additional correct mappings or they discovered correct mappings, which already exist in input mappings. It is understandable because almost structural methods compute similarity between two entities by determining how much overlap (e.g. Jaccard measure) of their structural patterns (i.e.m adjacent, ancestor, etc.). The higher filter threshold is, the lower possibility to discover new mappings is.

On the contrary, the matching quality of structural methods are significantly different when threshold value is small. When the threshold is set to very small value (from 0.01 to 0.09), ASCOPATH and ANCESTORS provide low matching quality. It means that these methods discover many incorrect mappings. For example, when the threshold is equal to 0.01, ACSOPATH discovers 90 (4733 - 4643) additional correct mappings but 453 (480 - 27) incorrect mappings in comparison with init mappings. It can be explained as follows. Due to observation of ontologies in Benchmark 2011 dataset, we see that the maximum depth and also the maximum number of ancestors of an entity in the ontology hierarchy is equal to 5. Assume that two entities have only one common entity in their ancestors, then their similarity score at least is equal to $1/10 = 0.1$. If two entities do not have any common entity, then their similarity is equal to 0. Therefore, with threshold in range from 0.01 to 0.09, any pair of entities having at least one common ancestor will be assumed as matched. Since siblings entities have the same path and ancestors, they will have the same structural patterns. Therefore, many pairs of entities have the same similarity scores. Moreover, one entity may have many descendant entities so many pairs of entities can be coupled, consequently, many incorrect mappings are produced.

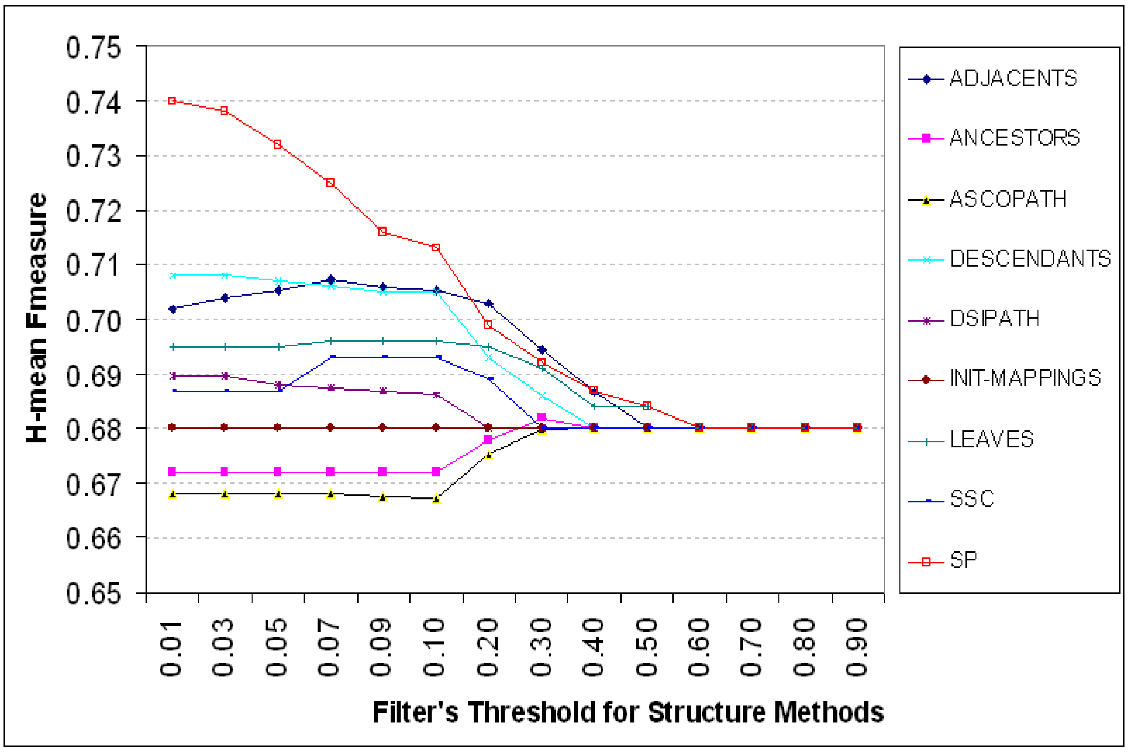


Figure 4.8: Comparison of different matching methods in structure based matcher

Whereas, other methods such as DESCENDANTS, SSC, DSIPATH and LEAVES seem to work well with small thresholds. They discover more additional correct mappings than incorrect mappings and, consequently, they improve the quality of matching. For example, with threshold is equal to 0.01, DESCENDANTS discovers 494 = (5137 - 4643) additional correct mappings and 175 = (202 -27) incorrect mappings in comparison with init mappings. Similar to ASCOPATH and ANCESTORS methods, with low threshold filter, many pairs of entities are passed. However, these methods clearly distinguish the structural patterns of entities. For instance, in DESCENDANTS and LEAVES, different entities have different sets of leaves/ descendants; in DSIPATH and SSC, they use different contribution percentage of entities according to how much an entity is important to another [92]. Therefore, by running greedy selection, high percentage of selected mappings are correct.

Our proposed Similarity Propagation (SP) is different with these structural methods discussed above. Note that the similarity scores produced by SP is not the absolute but relative values due to normalized process at the end of each running iteration. SP propagates similarity values from one pair of entities to others, hence, if two entities have similarity score higher than 0, then they are somehow similar. Thus, with a low threshold filter, SP discovers more correct mappings than that with a high threshold value. Moreover, similarity score of a pair of entities depends on not only their current status but also on the status of the other pairs. The more neighbors with high similarity a pair of entities have, the higher possibility that they are matched. Therefore, SP distinguishes well correct and incorrect mappings by ranking similarity scores. That explains why SP outperforms all other local based structural methods discussed above when the filter threshold is low. For example, when the threshold is equal to 0.01, SP discovers additional 1298 (5941 - 4643) correct mappings and 247 (274 - 27) incorrect mappings in comparison with init mappings. It shows that SP produces better matching quality result than other methods in the structure based matcher module.

## 4.4.2 Impact of Input Noise on the Structure-based Methods.

In this experiment, we evaluate the behavior of different structure methods when we add noise data to input mappings. Here, we call noise a pair of dissimilar entities but discovered as similar by element based matcher. It is important because in real scenario matching case, a matching method rarely produces 100% precision, consequently, it rarely provides input mappings without noise to structure methods. Intuitively, when noise data increase, the number of incorrect mappings increases whereas, the number of correct mappings decreases. Our assumption is that a stable method will produce less incorrect mappings than correct mappings. Therefore, we will study the changes of the number of correct and incorrect mappings discovered by each structure method. The evaluation strategy works as follows:

- At Element based matcher, we use Identical similarity measure to produce initial mappings. In order to make noise, we add a number of random incorrect mappings to inputs, which is corresponding to N% of size of original init mappings. Here N = (0,10,..,100).
- At Structure based matcher, SP takes input mappings from Element based matcher and performs similarity propagation. According to the experiment in

section 4.4.1, we select the best filter threshold value for each structure method. For example, $\theta_{SP} = 0.01$, $\theta_{DESCENDANTS} = 0.01$, $\theta_{ADJACENTS} = 0.07$, etc.

- For each running, we count the total number of correct mappings and the total number of incorrect mappings that a structure method produces overall 103 test cases in Benchmark 2011 dataset.

Fig. 4.9 shows the total number of correct and incorrect mappings produced by the structure methods for each time noise data are added to inputs. Generally, when more noise data are added, the number of correct mappings discovered by all the methods decreases ,whereas, the number of incorrect mappings discovered by almost methods increases except DSIPATH and SSC. Here, DSIPATH and SSC are unlike other local based structure methods in terms of interaction between entities in ontology. For example, the similarity of two entities computed by DSIPATH strongly depends on their similarity provided by input mappings and decreasingly depends on similarity of parents, grandparents, etc. Consider two entities of two input ontologies. If noise appears at the same level in their path to root, their similarity will be impacted by noise, otherwise, it will not. Therefore, the impact of noise in discovering others mappings depends on the position of its entities in the hierarchies of input ontologies. Because noise data are created randomly, the impact of noise to produce incorrect mappings is unpredictable. Whereas, other structure methods use set operations (i.e. intersection, union), so there is no difference between parent and grandparent. When a noise appears in the set of ancestors or descendants of two entities, the noise will directly propagate errors to them. Therefore, obviously in Fig. 4.9, the number of incorrect mappings increases in almost all structure methods.

This experiment also shows the dominant of using Similarity propagation over other structure methods. Let's see on the diagram representing the number of correct mappings discovered in Fig. 4.9. When the percentage of noisy data is 100%, SP still discovers 913 additional correct mappings in comparison with init mappings. Whereas, the maximum number of correct mappings discovered by the other methods is only 612 mappings when there is no noise added to the inputs. Moreover, in the next diagram in Fig. 4.9, from 0% to 100% of the noisy data, SP produces only 57 (321 - 274) additional incorrect mappings. Whereas, for example, LEAVES method produces 481 (553 - 72) more inccorect mappings. This feature is reasonable because SP takes all kinds of semantic relations of entities such as concept-concept, concept-property and property-property into account. These constraint relations will reduce the impact of the noisy data to produce mappings. That is why SP is

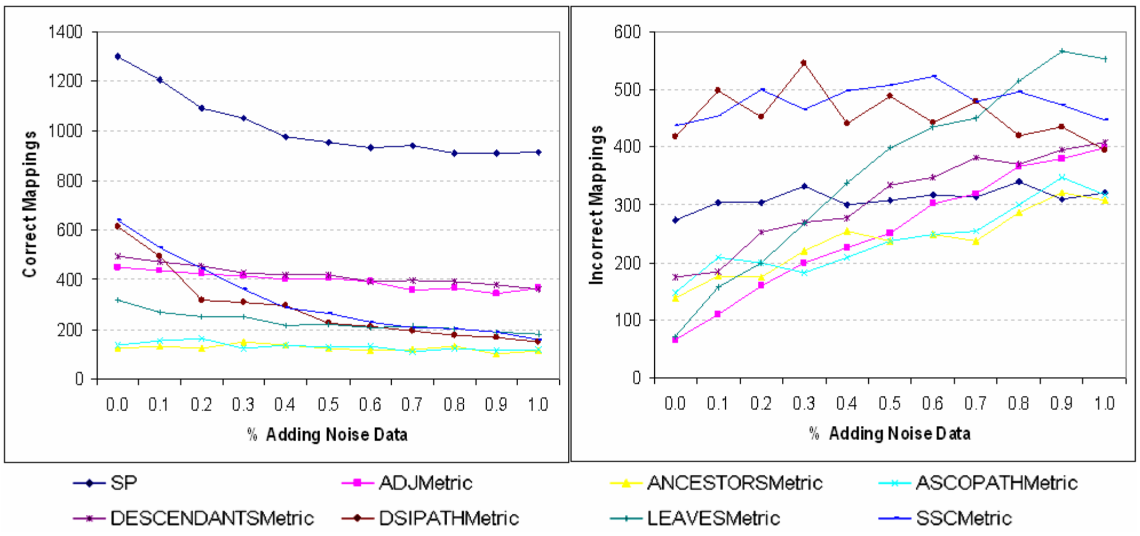
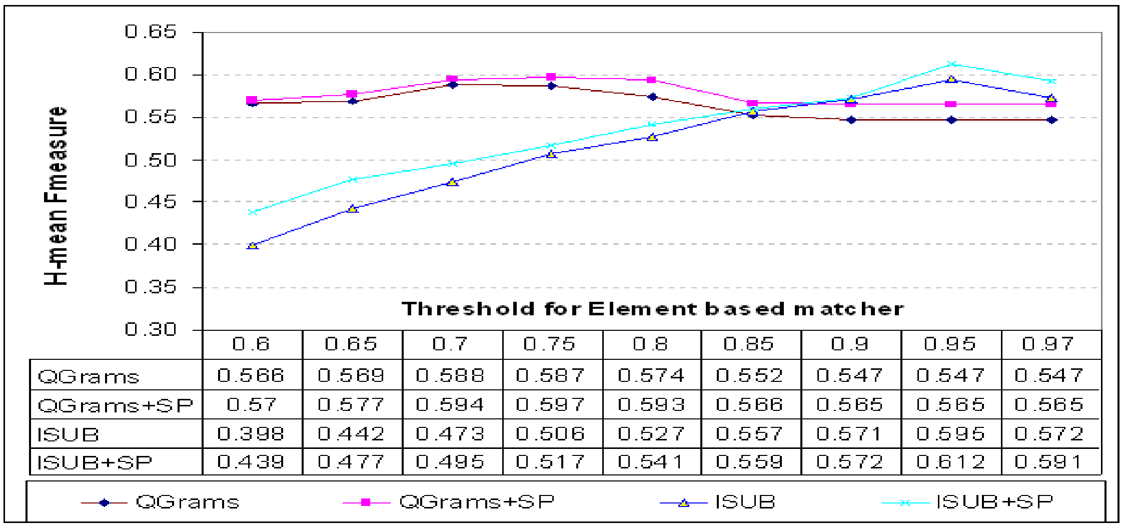


Figure 4.9: Impact of noise input to structure based methods

known as a stability constraint method.

### 4.4.3 Impact of the Quality of Input to Similarity Propagation Method

In this experiment, we investigate the inpact of input on the Similarity Propagation (SP) method. To do that, we select a string based matcher to produce input (initial mappings) to the SP process. From the Fig. 3.3, we choose QGRams and ISUB matchers because they show different manner when the threshold used to select the mappings at element level changes.



| | 0.6 | 0.65 | 0.7 | 0.75 | 0.8 | 0.85 | 0.9 | 0.95 | 0.97 |
|---|---|---|---|---|---|---|---|---|---|
| QGrams | 0.566 | 0.569 | 0.588 | 0.587 | 0.574 | 0.552 | 0.547 | 0.547 | 0.547 |
| QGrams+SP | 0.57 | 0.577 | 0.594 | 0.597 | 0.593 | 0.566 | 0.565 | 0.565 | 0.565 |
| ISUB | 0.398 | 0.442 | 0.473 | 0.506 | 0.527 | 0.557 | 0.571 | 0.595 | 0.572 |
| ISUB+SP | 0.439 | 0.477 | 0.495 | 0.517 | 0.541 | 0.559 | 0.572 | 0.612 | 0.591 |

Figure 4.10: Impact of the quality of input to the similarity propagation method

Fig. 4.10 shows the changes of matching quality obtained by SP when the init mappings changed. Obviously, line ISUB increases from 0.398 to 0.595, then line ISUB+SP increases from 0.439 to 0.612. Both lines QGrams and QGrams+SP go up to the peak (threshold is equal 0.7) then go down. Therefore, when the matching quality of the inputs improves, the matching quality of the SP increases too. According to this experiment, we may conclude that the better initial mappings provided to the similarity propagation are, the better matching result will be obtained.

### 4.4.4 Impact of Using a Reasoning System to the Similarity Propagation Method

In this section, we will show the advantage of using a description logic reasoner in the ontology matching task. Apparently, the ontology reasoner detects the hidden relations between entities; consequently, new edges in ontology graph will be added. Therefore, it impacts to the result obtained by propagation process in structural method. To see that, we configure our method running with and without a reasoner on some test cases. In this experiment, we use Pellet as a description logic reasoner.

| `Configuration` | `Pr.` | `Re.` | `Fm.` | `TP` | `FP` | `FN` |
|---|---|---|---|---|---|---|
| `with Pellet` | 0.979 | 0.609 | 0.751 | 5470 | 115 | 3515 |
| `no Pellet` | 0.978 | 0.608 | 0.749 | 5462 | 120 | 3523 |

Table 4.3: Comparison of matching quality obtained with OAEI Benchmark 2011 dataset

| `Configuration` | `Pr.` | `Re.` | `Fm.` | `TP` | `FP` | `FN` |
|---|---|---|---|---|---|---|
| `with Pellet` | 0.750 | 0.570 | 0.648 | 174 | 58 | 131 |
| `no Pellet` | 0.715 | 0.551 | 0.622 | 168 | 67 | 137 |

Table 4.4: Comparison of matching quality obtained with Conference dataset

We perform the two following experiments to see the comparison of matching quality obtained by running matching with two configurations, i.e., with and without Pellet reasoner.

- In the first experiment, we select OAEI Benchmark 2011 as test matching scenarios. Similar to section 4.4.1, we use Identical similarity measure to provide initial mappings to the similarity propagation process. Table 4.3 shows the comparison result of this experiment.

- In the second experiment, we select the real world ontologies in Conference dataset. The Identical similarity measure is used to provide initial mappings to the similarity propagation. Table 4.4 shows the comparison result of the average of of this experiment.

In the both experiments, using a reasoner discovers more correct mappings and less incorrect mappings. For example, in the Conference track, the number of correct mappings increases 6 ($174 - 168$) and the number of incorrect mappings decreases 9 ($67 - 58$). Therefore, in term of H-mean Fmeasure, using Pellet improves its value with 2.6%. Similarly, in the Benchmark OAEI 2011 track, the number of correct mappings increases 8 ($5470 - 5462$), the number of incorrect mappings decreases 5 ($120 - 115$) and $Fmeasure$ slightly increases by 0.002%. According to this experiment, we may conclude that the using a reasoning system slightly increases the matching quality.

## 4.5 Conclusion

In this chapter, we have presented our approach to deal with conceptual heterogeneity in ontology matching. For this purpose, we have analyzed different types of conceptual heterogeneity as well as the challenges in dealing with them. In addition, we have also discussed the ability of basic structural similarity measures when dealing with conceptual heterogeneity. We have concluded that the basic methods are incomplete and uncertain.

To overcome the challenging issues of the conceptual heterogeneity, we have proposed a modification of the Similarity Flooding algorithm [67], that is specifically adapted to the ontology matching task. Our method is called Similarity Propagation. It is based on a iterative computation, in which, the similarity of a pair of entities propagates to the other neighboring pairs in each iteration. The modification lies on the new graph data structure that we used for storing semantic information of an ontology. Thanks to the high level graph data structure, the semantic information encoded in the ontology becomes more visible. Consequently, the similarity propagation process can be easily implemented on it.

Our experiments have proved the importance of the similarity propagation method. Firstly, we have shown that this method outperforms other basic structural methods in terms of matching quality. It is because this method exploits more structural information than the others. Next, according to the experiment with noise input, we have concluded that the similarity propagation method is the most stable in comparison to the others. This property has been again verified in the experiments in section 4.4.3. Indeed, its performance correlates with the quality of the input. This method always improves the matching quality by discovering new correct mappings.

According to these experiments, the most important observation is that if we would obtain a high matching quality result by using this method, we should provide a good matching quality of input. Finally, using a reasoning system to build a high level graph from an ontology slightly improves the matching quality. However, because new hidden relations are added into the graph, the computational space becomes bigger, which leads to a requirement of big memory and high runtime computation.

In Chapter 3 and in the current chapter, we have discussed many similarity measures that can deal with terminological and conceptual heterogeneity. Each of the proposed measures has been tested in our experiments. The next question is how to combine these measures effectively. In the ontology matching field, this problem is known as the selection and configuration tuning challenge [76]. We will discuss and propose a solution to this issue in next chapter.

# Chapter 5

# Matcher Combination

The combination of different individual matchers is necessary and pervasive in ontology matching systems. In the ideal combination, the individual matchers should complement each other in order to increase the matching evidence of pairs of entities. Moreover, a clear distinction between correct and incorrect mappings should be made. However, in reality, designing an intelligent combination method like that is a real challenge in the ontology matching field. Two major problems arise: (i) selecting matchers and combining them, and (ii) self-configuring or tuning matchers [76].

So far, we have reviewed many individual matchers dealing with heterogeneity of ontologies. Each of these matchers is based on a similarity measure and exploits a specific feature of the entities. In particular, terminological matchers are based on terminological similarity measures, which compute a similarity value for two entities by comparing their terminological features – labels, comments, context, etc. On the other hand, structural matchers are based on structural similarity measures, which exploit the relation between entities for the similarity computation. As we discussed in Chapter 3, the existing similarity measures including the ones proposed in this thesis can be sufficient for few types of terminological heterogeneity, but not for all. Thus, several methods should be combined together in order to effectively overcome the terminological heterogeneity. Moreover, since the semantics of an entity encompasses both terminological and structural information, using only one matcher may be not sufficient. Therefore, a combination between terminological and structural matchers is a requisite.

In this chapter, we propose our approach to deal with the challenge of matcher combination. For this purpose, in Section 5.1, we review first several automatic combination methods in the state-of-the-art ontology matching systems. In particular, two automatic combination methods, namely Harmonic Adaptive Weighted Sum

[64] and Local Confidence Weighted Sum [15] will be discussed in detail to underline their strengths and weaknesses.

Our first contribution contained in this chapter is a machine learning-based method, which is used to combine different terminological similarity measures (Section 5.2). The intuition behind this method is that an ontology matching task can be transformed into a task of a classification of objects, which can be solved by using machine learning models. Furthermore, 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 5.3). For a given matching scenario, this method evaluates the degree of reliability of these matchers, and assigns a corresponding weight values to them. In addition, it also automatically determines a threshold value to select a combined matching result.

Finally, the performance of our proposed methods will be evaluated in experiments described in Section 5.4. In these experiments, we prove that our methods outperform Harmonic Adaptive Weighted Sum and Local Confidence Weighted Sum methods.

## 5.1 Overview of Automatic Combination Methods

In [28], many combination methods have been proposed to aggregate similarity values of different individual matchers. For example, the **Max/Min** methods return 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 function 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.

In addition **Max/Min** and **Average** methods do not require any parameter settings but they are limited to use in some specific circumstances. In particular, the **Average** method assumes that no one of the individual matchers is more important than the others. But, in practice, some entities in the input ontologies can work with some but not all individual matchers. For example, some entities have properties, they can work with matchers based on similarity of properties. Whereas the other entities do not have properties, thus they cannot work with those matchers. This problem can solved by using the **Max/Min** methods. However, these methods assume that a mapping can be completely discovered by using only one extracted features of entities. For example, some mappings can be found by comparing names of entities only, whereas, the others may be found by comparing labels or descriptions of entities, etc. That is, the certainty of the **Max/Min** methods strongly depend on the certainty of the individual matchers. Therefore, these methods can be useful when the individual matchers are strong and high certainty.

To our best knowledge, in recent years, 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 analyses in [64], this method outperforms all the methods mentioned above. The second method is *Local Confidence Weighted Sum* [15]. It is a core method for combining individual matchers in the AgreementMaker system which is one of the top best ontology matching system in recent OAEI campaign. The brief overview of the two methods will be discussed in the next sections.

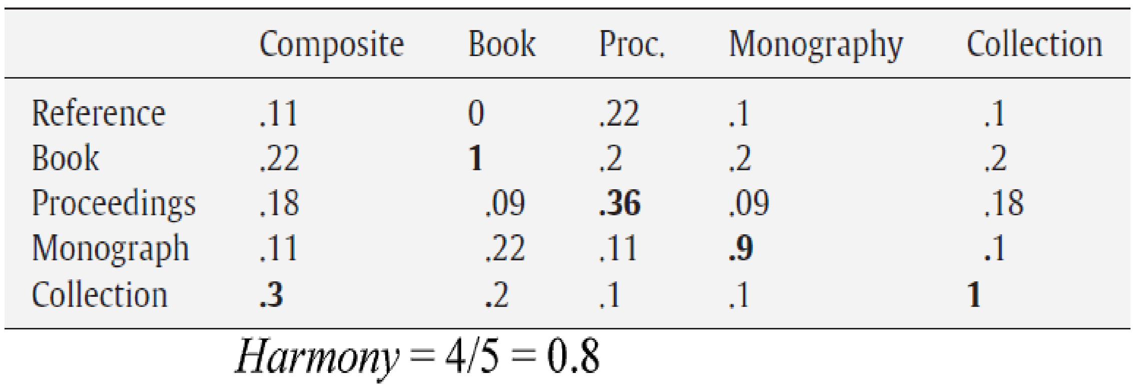
### 5.1.1 Harmonic Adaptive Weighted Sum Method - HW

This method is based on the notion of *harmony* which estimates the importance and reliability of different individual matchers. In order to determine a harmony value for a given individual matcher, this method assumes that the matching cardinality is 1:1. The intuition of this method is as follows. The similarity value of two truly mapped entities (e.g., $\langle a_i, b_j \rangle$) should be larger than that of all other pairs of entities that contain either $a_i$ or $b_j$. It implies that the two entities $a_i$ and $b_j$ mutually prefer each other.

Assume that a given individual matcher produces a similarity matrix for all pairs of entities from the two input ontologies. The harmony value of a given individual matcher with two input ontologies $O_1$ and $O_2$ is computed by the following formula:

$$harmony = \frac{\|perfectMatches\|}{\|O_1\| + \|O_2\|}$$

where, $perfectMatches$ is a set of the pair of entities that has the highest and the only highest similarity in its corresponding row and column in the similarity



| | Composite | Book | Proc. | Monography | Collection |
|---|---|---|---|---|---|
| Reference | .11 | 0 | .22 | .1 | .1 |
| Book | .22 | **1** | .2 | .2 | .2 |
| Proceedings | .18 | .09 | **.36** | .09 | .18 |
| Monograph | .11 | .22 | .11 | **.9** | .1 |
| Collection | **.3** | .2 | .1 | .1 | **1** |

$Harmony = 4/5 = 0.8$

Figure 5.1: An example of computing a harmony value (taken from [64])

matrix. Let see an example in the Fig.5.1, the harmony value is equal to $\frac{4}{5} = 0.8$.

Once harmony values are calculated for all individual matchers, the **harmonic adaptive weighted sum** method uses these values as weights for the individual matchers. Then, the final similarity value of two entities $(a_i, b_j)$ can be computed as follows:

$$finalSim(a_i, b_j) = \frac{\sum_k h_k \cdot Sim_k(a_i, b_j)}{n}$$

where, $n$ is the total number of individual matcher being combined; $h_k$ is a harmony value of the $k$th individual matcher, and $Sim_k(a_i, b_j)$ denotes the similarity value of two entities $(a_i, b_j)$ computed by the $k$th individual matcher.

### 5.1.2 Local Confidence Weighted Sum Method - LC

This method is based on the notion of *local confidence* measure, which estimates the degree of the reliability of a given individual matcher that discovers mappings for a given entity. Note that this measure is very different to the harmony measure discussed in above. In particular, a harmony value is a degree of reliability of similarity values of all pairs of entities computed by a matcher, whereas, each local confidence value is a degree of reliability of a similarity value of two specific entities computed by a matcher.

The computation of local confidence is based on the following intuition. Firstly, it should be directly proportional to the similarity values of selected mappings. Next, it should detect and penalize those matchers that tend to assign high similarity values too generously. For instance, if the matching cardinality is 1:1, a reliable matcher will discover each entity to be very similar (i.e., have high similarity value) to one entity at most, and very different (i.e., have low similarity value) to all others.

Based on this intuition, given a matcher $M$ and an entity $c$, the local confidence

$LC_M(c)$ of $M$ with respect to $c$ is computed as follows:

- Let $T$ be the set of all target entities;
- Let $m_M(c) \subseteq T$ be the set of concepts $c' \in T$ that have been mapped to $c$ by $M$;
- Let $sim_M(c, c')$ be the similarity value between $c$ and $c'$ assigned by $M$;
- Then $LC_M(c)$ is defined as the difference between the average of selected mappings similarities for $c$ and the average of the remaining correspondences' similarities:

$$LC_M(c) = \frac{\sum_{c' \in m_M(c)} sim_M(c, c')}{\|m_M(c)\|} - \frac{\sum_{c' \in T \backslash m_M(c)} sim_M(c, c')}{\|T \backslash m_M(c)\|}$$

Then, the final similarity value of two entities $(a_i, b_j)$ can be computed as follows:

$$finalSim(a_i, b_j) = \frac{\sum_k LC_k(a_i, b_j) \cdot Sim_k(a_i, b_j)}{\sum_k LC_k(a_i, b_j)}$$

where, $LC_k(a_i, b_j)$ is a local confidence value of the $k$th individual matcher with respect to pair of entities $(a_i, b_j)$; and $Sim_k(a_i, b_j)$ denotes the similarity value of two entities $(a_i, b_j)$ computed by the $k$th individual matcher.

### 5.1.3 Observation on the HW and LC methods

Generally, both HW and LC methods somehow solve the problem of assigning weights to individual matchers automatically. They both analyze the similarity values computed by a given matcher in order to estimate their degree of reliability. They are flexible because there is no restriction on the number of individual matchers being combined.

However, there are several weaknesses in these methods. Firstly, in some situations, different individual matchers are complementary, i.e., they can only handle a part of entities well, therefore with the HW method, all these matchers may be assigned a low reliability. In that case, the similarity values computed by individual matchers seem to be shrank into smaller range. Thus, the distinction between correct and incorrect mappings becomes slight. It makes a trouble in the selection of final alignment. On the other hand, this weakness of the HW method is somehow fixed in the LC method. It is because the LC method estimates the reliability of each pair of entities instead of the whole similarity matrix as the HW does.

Secondly, both methods may not really satisfy the natural aim of a weighted sum model, which emphasizes the contribution of the high reliable matchers and weakens the contribution of the low reliable ones. For example, in general, the similarity values obtained by a terminological similarity measure are usually higher than that obtained by a structural similarity measure. In that case, the local confidence of the first method is higher than the local confidence of the second method. It is a contradiction because the reliability of a terminological similarity measure is usually less important than the reliability of a structural similarity measure.

Finally, they both lack a strategy to determine a threshold value to select the best mappings. In fact, a reliability value of a given individual matcher varies with respect to different matching scenarios. Therefore, the final similarity values are dependent to the input ontologies in a given matching scenario. Thus, the filter threshold should be selected dynamically and dependent to the matching scenarios.

## 5.2 Machine Learning Based Combination Method - ML

In this section we present our approach for combining similarity measures, which is based on machine learning. The main idea is as follows. From existing gold standard dataset, a classification will be built. Here, a "gold standard" data is a pair of ontologies with an alignment provided by domain experts. Given a matching scenario, for each pair of entities in ontologies, the classification classifies them in to match or not match category. To do that, we are going to deal with the following issues.

The **first** issue is about *how to generate training data from gold standard dataset and how to generate unclassified object from two entities of the input ontologies?* In fact, the way of generating both training and unclassified data are the same. For each pair of entities, a list of similarity scores is computed by applying a list of similarity measures. Here, each pair of entities is considered as a learning object $X$; each similarity measure becomes a $X$'s attribute and its corresponding similarity score is considered as a $X$'s feature value. If two entities are in two to-be-matched ontologies, $X$ becomes an unclassified object. We set an unknown value to its class. If two entities are in ontologies within the gold standard dataset, $X$ becomes an instance of training data. Its class value is assigned by its confidence value found in expert alignment. In our approach, we categorize the class values into two groups which mean two entities are matched (value 1.0) or not matched (value 0.0).

The **second** issue is *which similarity measures will be selected for combination?*

Theoretically, all terminological similarity measures can be used as attributes, but it will make the learning and classifying processes be time consuming. In our approach, we select representative similarity measures that can deal with different types of terminological heterogeneity described in Section 3.1. In particular, to deal with type 1, we select **Levenstein** and **ISUB**, which are representative for a group of edit-based similarity measures; to deal with type 2, we select **QGrams** and **TokLev**, which is a token-based similarity measure that use Levenstein as its internal similarity measure. Next, in order to deal with type 3, we select two hybrid similarity measures, namely **HybLinISUB** and **HybJCLev**. Where, **HybLinISUB** (or HybJCLev) is hybrid similarity measure that combines **Lin** with **ISUB** measures (or **JiangCorath** and **Levenstein** measures). Their algorithms are described in Section 3.2. To deal with type 4 and type 5 of terminological heterogeneity, we make use of a **MaxContext** measure, which returns a maximum similarity of three types of context profiles (i.e., IndividualProfile, SemanticProfile and ExternalProfile) of entities. It is described in Section 3.3.2. To sum up, Table 5.1 shows the list of the selected similarity measures. Surely, we can add other similarity measures in combination; our method will automatically tune new parameters to combine them.

| Type | List of measures |
|---|---|
| Type 1 | Levenstein, ISUB |
| Type 2 | QGrams, TokLev |
| Type 3 | HybLinISUB, HybJCLev |
| Type 4-5 | MaxContext |

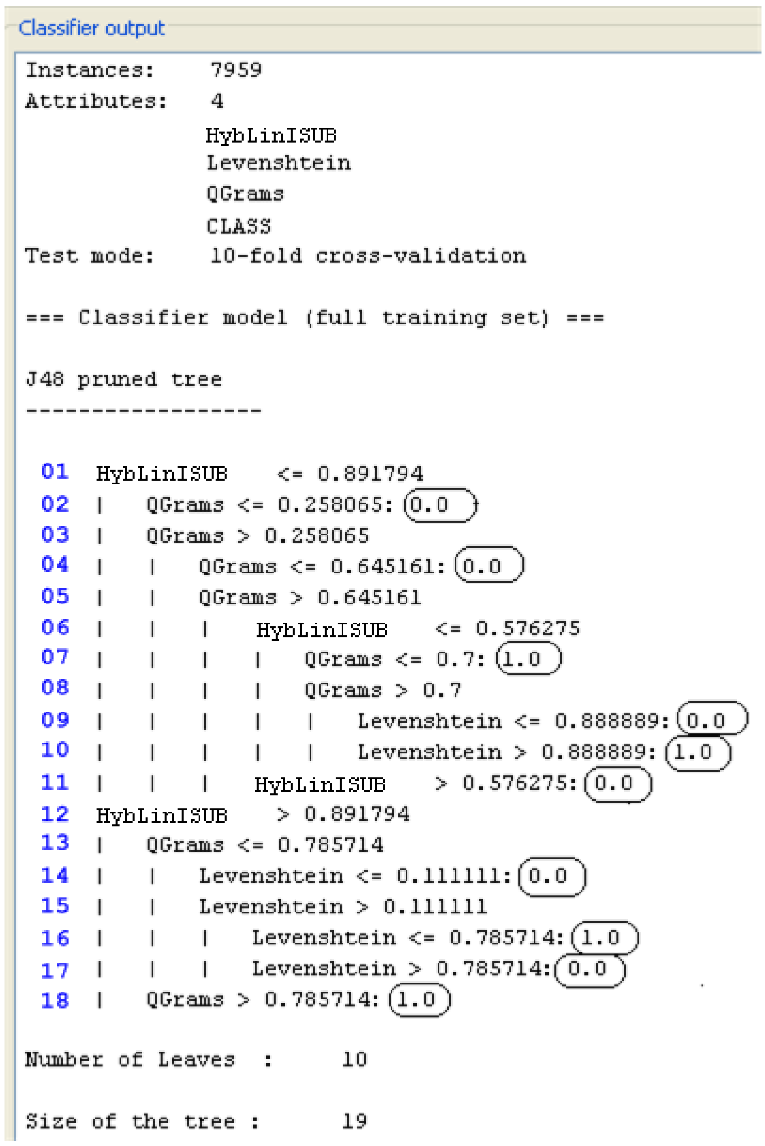
Table 5.1: List of the selected similarity measures

The **third** issue is about *what machine learning model will be used to build a classification from given training data?* Different machine learning models (e.g., tree-based, probability-based, function-based, rule-based, instance-based, etc.) can be used to build a classification. The implementations of these machine learning models are reused from the well-known open source data mining framework Weka[1].

The **last** issue is about *how to classify an unclassified object to its class in order to check if two entities corresponding to this object are match or not?* Let us demonstrate the classifying process with the motivating example described in Chapter 1. Assume that we use a decision tree model to combine the three similarity measures i.e., Levenstein, Qgrams, HybLinISUB. The "gold standard" dataset used to generate training data is taken from Benchmark OAEI 2009 track.

Fig. 5.2 shows the classification obtained after the training process. In this

[1]http://www.cs.waikato.ac.nz/ml/weka


Classifier output

```
Instances:     7959
Attributes:    4
               HybLinISUB
               Levenshtein
               QGrams
               CLASS
Test mode:     10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

 01  HybLinISUB    <= 0.891794
 02  |   QGrams <= 0.258065: 0.0
 03  |   QGrams > 0.258065
 04  |   |   QGrams <= 0.645161: 0.0
 05  |   |   QGrams > 0.645161
 06  |   |   |   HybLinISUB    <= 0.576275
 07  |   |   |   |   QGrams <= 0.7: 1.0
 08  |   |   |   |   QGrams > 0.7
 09  |   |   |   |   |   Levenshtein <= 0.888889: 0.0
 10  |   |   |   |   |   Levenshtein > 0.888889: 1.0
 11  |   |   |   HybLinISUB    > 0.576275: 0.0
 12  HybLinISUB    > 0.891794
 13  |   QGrams <= 0.785714
 14  |   |   Levenshtein <= 0.111111: 0.0
 15  |   |   Levenshtein > 0.111111
 16  |   |   |   Levenshtein <= 0.785714: 1.0
 17  |   |   |   Levenshtein > 0.785714: 0.0
 18  |   QGrams > 0.785714: 1.0

Number of Leaves  :      10

Size of the tree :       19
```

Figure 5.2: The trained decision tree classification

example, a decision tree is a tree whose non-leaf nodes are the similarity measures, leaf nodes values are either 1.0 or 0.0 indicating if there is a match or not. In Fig.5.2, leaves are represented by rounded rectangle shapes with number inside. At a non-leaf node, a similarity value of to-be-matched entities is computed by the similarity measure stored in this node. The returned value is compared with condition values on outgoing edges from current node in order to decide which child node will be reached. Here, all condition values are determined automatically by algorithm of building decision tree with given training data. The classification process will start at root node and iterate until a leaf node is reached. The value of destination leaf node indicates whether the two entities should match or not. In Fig. 5.2, edges with the condition values are indexed by numbers in pre-order traversal of 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 5.2: A set of the unclassified data

Now, we demonstrate how we use this decision tree classification in our system by several examples in Table 5.2. Here, we use `Hyb.`, `Lev.`, `QGs` and `CLS` as abbreviation of `HybLinISUB`, `Levenshtein`, `QGrams` and `CLASS` attributes respectively.

Let us see 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 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 from the training process at the root node. Therefore, the decision goes through the first edge (`HybLinISUB` <= `0.891794`). In the next node, QGrams, the returned similarity score is 0.80, which is higher than condition value `0.258065`. Therefore, the decision goes through the edge number 03. Similarly, this score is higher than 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 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 edge number 08 to the `Levenshtein` node. Since the similarity score returned by `Levenshtein` measure is `0.91` higher than condition value `0.888889`, therefore the decision reach to leaf with label `1.0` on edge number `10`. It means that entities `Researcher` and `Reseacheur` are matched. To sum up, edges on the path of the decision for those two entities is: `01` → `03` → `05` → `06` → `08` → `10` → `leaf(1.0)`.

In the same way, we can make the decision for the rest in unclassified data as follows. The decision path for the second instance on Table 5.2 is: `01` → `02` → `leaf(0.0)`. The decision paths for the third and fourth instances on Table 5.2 are the same as: `12` → `13` → `15` → `16` → `leaf(1.0)`. It means two entities `Teacher` and `Lecturer` are not match, whereas, `Manager` matches to `Director` and `teach` matches to `teaching`. The full results obtained by using this decision tree are shown in the Table 5.3.

| 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 5.3: Classified mappings by the trained decision tree

## 5.3 Dynamic Weighted Sum Method - DWS

In this section we will present our method to combine the mapping results obtained from a terminological matcher (or an element matcher in general) and a structural matcher. The main idea of this method is explained in Algorithm 3. Here, $A_{element}$ is a set of mappings discovered by the terminological matcher. $A_{structure}$ is the set of mappings discovered by the structural matcher. The similarity values $c_e$ and $c_s$ computed by these matcher are in range $[0, 1]$.

```
Algorithm 3: Produce Final Mappings
  input  : $A_{element} = \{(e_i, e_j, \equiv, c_e\}$
           $A_{structure} = \{(e_p, e_q, \equiv, c_s)\}$
  output: $A_{final} = \{(e_1, e_2, \equiv, c)\}$
1 $\theta \leftarrow \texttt{min}(m.c_s) \,|\, m \in A_{structure} \cap A_{element}$;
2 $A \leftarrow \texttt{WeightedSum}(A_{element}, \theta, A_{structure}, (1-\theta))$;
3 threshold $\leftarrow \theta$;
4 $A_{final} \leftarrow \texttt{GreedySelection}(A, \text{threshold})$;
5 return $A_{final}$
```

To take the contribution of both terminological and structural matchers into account, we use a weighted sum method to combine them. In order to avoid manual setting, this method should automatically set weights to element and structure matchers and select a threshold to filter mappings (lines 1,2 and 3 in Algorithm 3). Let us explain our method in Fig.5.3. Here, $M_{element}$ is a set of mappings discovered by only the terminological matcher. Similarly, $M_{structure}$ is a set of mappings discovered by only the structural matcher. They are indicated by labels with pink "em" and light-blue "sm" prefixes respectively. $M_{overlap}$ is a set of mappings discovered by both terminological and structural matchers. They are labeled with yellow "se" prefix.

In Fig.5.3, obviously, the mappings belong to $M_{overlap}$ = {se1,se2,se3} are the most potentially matched because their entities seem to have both similar

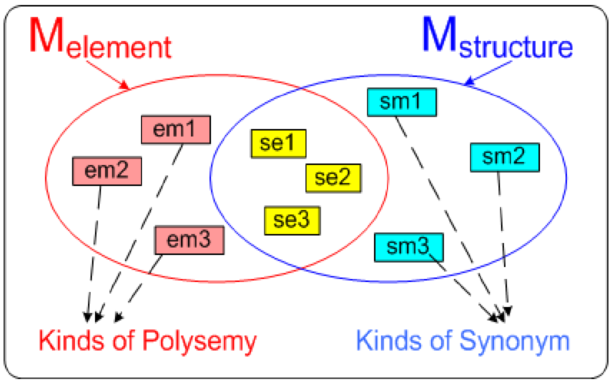


Figure 5.3: Four kinds of candidate mappings

name/labels and similar semantic description. Next, the mappings belong to $M_{structure}$ = {`sm1`,`sm2`,`sm3`} are kind of synonym because their entities seem to have different name/labels but have similar semantic descriptions. Whereas, each mapping belong to $M_{element}$ = {`em1`,`em2`,`em3`} is kind of polysemy because their entities seem to have similar name/ labels but different semantic descriptions. Intuitively, the explicit meaning of entity (through 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 mappings in $M_{overlap}$ are correct mappings.

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

Let us see Algorithm 3 to understand how we calculate the confidence value for mappings in $M_{element}$ and filter threshold for mappings in $M_{structure}$. Firstly, we seek the minimum value of structural similarity score in $M_{overlap}$ (line 1). We assume that all mappings having a structural similarity score, which is higher than this value will be considered as correct. Therefore, we select this value as filter threshold $\theta$. According to our intuition discussed above, the possibility of correctness of mappings in $M_{element}$ is smaller than priority of mappings in $M_{structure}$, we will set the confidence to the mappings in $M_{element}$ to $\theta$. 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 element and structure levels to $\theta$ and $1 - \theta$ respectively. Then we compute their similarity by weighted sum function (line 2).

For the illustration of this idea, we continue with the motivating example described in the Introduction chapter. The terminological matcher is based on machine learning models to combine different terminological similarity measures described in the previous section. The structural matcher is based on the Similarity propagation method described in section 4.3. Indeed, the input of the structural matcher is taken from the output of the terminological matcher. By running the Similarity propagation on the input ontologies, we obtain the following result in Table 5.4.

| 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 5.4: Discovered mappings by similarity propagation method

| 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 5.5: Combination results of 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 5.6: Result after greedy selection. $\theta = 0.2716023$

Let us see mapping results in Table 5.3 and Table 5.4. There, pair of entities (`Employee`, `Employee`) has the minimum value ($c_s = 0.271$) found in overlap between results of element level and structure level matchers. Then, we set $\theta = 0.271$. Table 5.5 shows the combination results obtained from element level and structure level matchers. After perform a Greedy Filtering with threshold $\theta = 0.271$, the final result is shown in Table 5.6.

## 5.4 Experiments and Evaluations

In this section, we design 4 experiments to evaluate the performance of our proposed combination methods. In the first experiment (Section 5.4.1), we compare the performance of different machine learning models that can be used to build a classification for the ontology matching task. In the second experiment (Section 5.4.2), we investigate the impact of selected similarity measures on the performance of the machine learning based combination method. Next, in Section 5.4.3, we compare the performance of the machine learning based method with two automatic weighted sum methods HW and LC. Finally, in Section 5.4.4, we compare the performance of our dynamic weighted sum method with HW and LC.

### 5.4.1 Comparison of Performance of Different ML Models

The aim of this experiment is to find the most suitable ML model for our approach. In fact, we can use different machine learning algorithms in order to build a classification model. These algorithms are divided in 5 groups as follows:

- Tree-based: J48, J48Graft, ADTree, SimpleCart, NBTree.
- Probability-based: NaiveBayes, BayesNet.

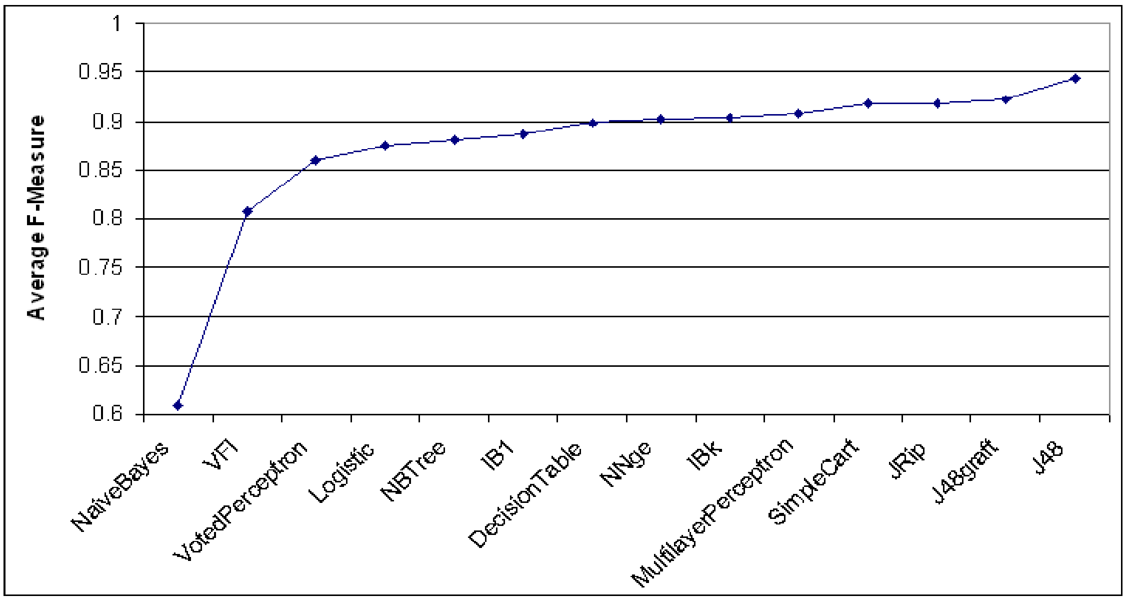


Figure 5.4: Comparison of the performance of learning models

- Function-based: Logistic, MultiLayerPerceptron.
- Rule-based: JRip, VFI, DecisionTable.
- Instance-based: IBk, NNGe.

To compare different models, we randomly select a set of "gold standard" datasets to generate training data and then measure the performance of each learning model by applying 10-fold cross-validation technique [100]. This process is repeated 10 times in order to limit the impact of randomness during the evaluation. The average of F-measure values of all learning models are displayed in Fig.5.4. The model which has the highest performance is J48 - a modified version of the decision tree model. Following the J48 model are J48graft, JRip and SimpleCart models. According to the comparison result, hereafter, we use J48 model in the learning and classification tasks.

### 5.4.2 Impact of Selected Similarity Measures on Performance of ML

In order to study the impact on the matching quality of similarity measures selected from 4 groups discussed above, we set up 4 different sets of selected measures used in combination by the decision tree model as follows:

- C1: Only string-based measures in Table 5.1, i.e., Levenstein, ISUB, QGrams, TokLev.

- C2: Collection of string-based and context profile measure, i.e., {S1} and MaxContext.
- C3: Collection of string-based and language-based measures, i.e., {S1} and HybLinISUB, HybWPLev.
- C4: All measures in Table 5.1.

In this experiment, we again select test cases from Conference track. To provide training data to build decision tree classifier, we use gold standard data sets from OAEI Benchmark 2009 and FOAM project. We randomly generate 10 different training data (numbered from 01 to 10). For each training data, we build 4 different decision tree classifications based on 4 different collections of selected similarity measures described above. We call these classifications as ML1, ML2, ML3 and ML4 respectively. These classifications are used to produce matching results for each test case in Conference track.

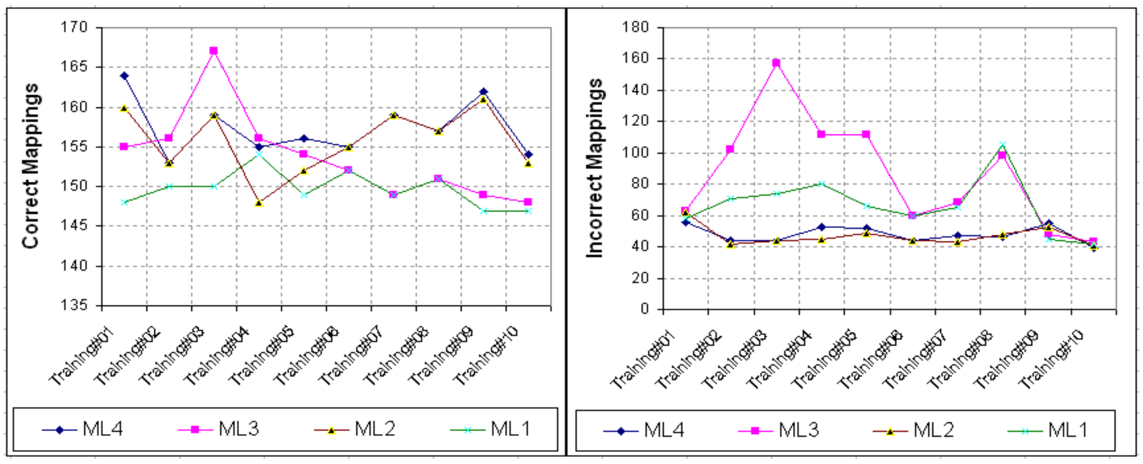


Figure 5.5: Comparison of combination on different collections of similarity measures

| | ML1 | ML2 | ML3 | ML4 |
|---|---|---|---|---|
| Average Fmeasure | 0.579 | 0.615 | 0.573 | 0.616 |

Table 5.7: Average of Fmeasure obtained by using running ML with 4 different collections of similarity measures

Here, we are going to study the impact of similarity measures on discovering both correct and incorrect mappings. Therefore, for saving space, we show only the total number of correct and incorrect mappings discovered by different combination of similarity measures overall test cases in Conference dataset in Fig.5.5. The observation from the Fig.5.5 is as follows:

- Firstly, the context profile measure seems to not only discover correct mappings but also to reduce incorrect mappings. Let see ML2 and ML1 in the Fig.5.5. Note that C2 consists of all measures in C1 and context profile similarity measure. In almost training data, the total number of correct mappings that ML2 discovered is higher than that ML1 did. Moreover, the total number of incorrect mappings that ML2 produced is smaller than that ML1 did. For example, by using context profile similarity measure, ML2 discovered concepts `PaperAbstract` and `Abstract` from `cmt.owl` and `conference.owl` respectively are matched, whereas ML1 did not. Moreover, ML1 discovered `Chairman` and `Chair_PC` from `cmt.owl` and `confOf.owl` matched but ML2 filtered this incorrect mapping.

- Secondly, the language-based measures discover not only additional correct mappings but also additional incorrect mappings. For example, by using language-based measures, ML3 discovered concepts `Chairman` and `Chair` from `cmt.owl` and `conference.owl` are matched because two labels are synonym indeed. Moreover, because the language-based measures remove all stop words, ML3 discovered `name`, `has_a_name` and `hasName` from `cmt.owl`, `conference.owl` and `edas.owl` respectively are matched also, but in fact they are incorrect mappings. Let see ML3 and ML1 in the Fig.5.5. In almost training data, the total numbers of correct and incorrect mappings that ML3 discovered are higher than that ML1 did. Additionally, Table 5.7 shows that the average of H-mean Fmeasure of ML3 (0.573) is smaller than the average of H-mean Fmeasure of ML1 (0.579). It means that using linguistic measures does not improve matching quality of system. Therefore, to take advantage of using linguistic measures effectively, we need to use another measure to remove the incorrect mappings produced by linguistic measures.

- Finally, Table 5.7 shows that ML4 is the best combination with average H-mean Fmeasure is equal to 0.616. ML4 is slightly better than ML2 (0.615). Note that C4 consists of all measures of C2 and addition linguistic measures. Thanks to linguistic measures, ML4 discovered more correct mappings. Besides, incorrect mappings produced by the linguistic measures were removed by the context profile similarity measure. For example, ML1 did not discover `name`, `has_a_name` and `hasName` from `cmt.owl`, `conference.owl` and `edas.owl` respectively are matched.

The experiment shows that using the combination of all selected similarity from the Table 5.1 produces the higher matching quality than other subsets of measures.

Therefore, by default we use all these similarity measures within the decision tree J48 model.

### 5.4.3 Comparison of Performance of ML, HW and LC

In this experiment, we compare the matching performance of **ML**, **LC** and **HW** on the Conference[2] dataset containing 16 real world ontologies describing the conference organization domain. These ontologies were developed by different people, concepts' labels are highly heterogeneous. Therefore, we assume that if a matcher obtains a high matching quality over this dataset, it would perform well over other real matching scenarios.

In the experiment, similarity measures in Table 5.1 are selected as individual matchers being combined by **ML**, **LC** and **HW** methods. To build a classification in **ML**, we use gold standard data sets from Benchmark and FOAM project as training data to make sure that training and testing data are independent. Here, we performed 10 times with different gold standard datasets in order to have different training data and to limit the impact of randomness during the evaluation. Then, we sort H-mean values overall 10 executions in an ascending order. Note that by using **ML** method, we do not need to set a threshold for selecting mappings. Whereas, for each individual matcher and for LC, HW methods discussed above, we need to set a filter threshold to select mappings. The threshold value $\theta$ is tuned from `0.6` to `0.97`.

Obviously, the ML combination method performed better than the others including individual matchers and LC and HW combination methods. The average Fmeasure of ML (`0.60`) is higher than the maximum value Fmeasure of HW (`0.56` at $\theta = 0.75$), LC (`0.55` at $\theta = 0.80$). ML is better than HW and LC because it does not use linear arithmetic functions to combine individual matchers, instead, it extracts the combination rules and constraints between them from training data. For example, ML method discovers (Co-author $\equiv$ Contribution_co-author) in ontologies `cmt.owl` and `conference.owl` respectively. It is because ML finds many patterns in training data similar to the current example (e.g., (networkA.rdf# Office $\equiv$ networkB# OfficeSoftware), (russia1# payment $\equiv$ russia2# means_of_payment), etc.). , Whereas, individual matchers and their combination by HW and LC return low similarity score between two labels, for instance, Levenstein(Co-author, Contribution _co-author) = 0.4; QGrams(Co-author, Contribution_co-author) = 0.6; LC(Co-author, Contribution _co-author) = 0.57.

[2]http://oaei.ontologymatching.org/2011/conference/

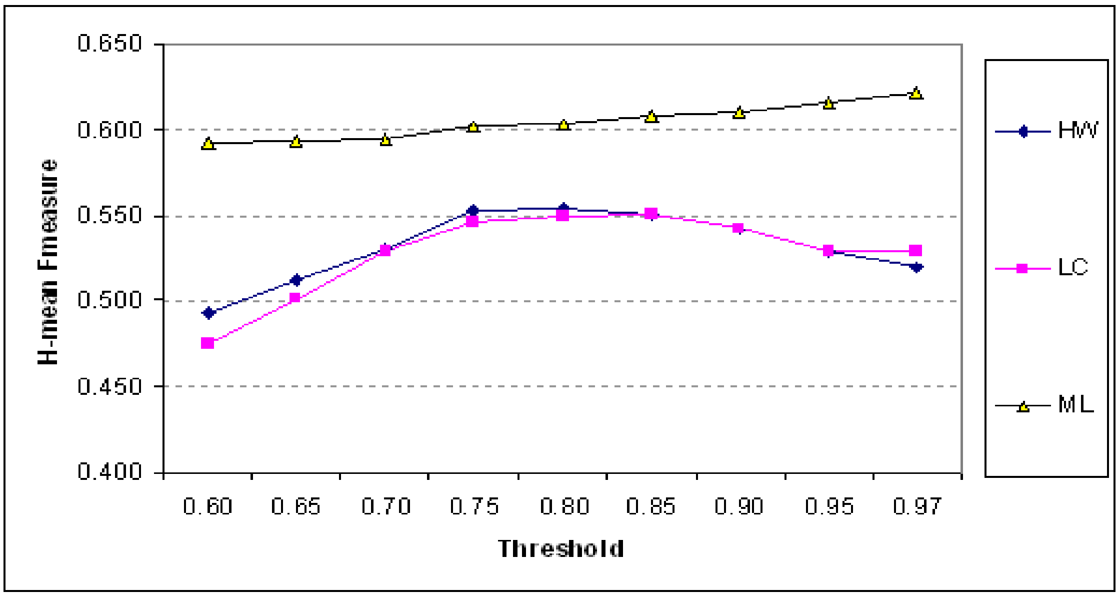


Figure 5.6: Comparison of Fmeasure of different combination methods on Conference dataset

Apparently, in Fig. 5.6, we can see that HW and LC have their own threshold that provides the best performance. For example, with HW, we will select threshold equal to 0.75; with LC we select 0.85. Because there is no guarantee that when threshold gets high, the Fmeasure improves; the selection of a good threshold is really a challenge to all matchers. An advantage of ML method is that it does not require any filter's threshold value.

Moreover, our method is flexible because there is no limit to add new individual matchers (aka. similarity measures) to the combination. Furthermore, it frees the user from the effort of setting filter's threshold to select final mappings.

### 5.4.4 Comparison of Performance of DWS, HW and LC

In this experiment, we compare the performance of our method DWS with HW and LC methods in combining matching results of an element matcher and a structural matcher. The element matcher is a combination of terminological similarity measures by using a machine learning model (**ML**). The structural matcher is based on similarity propagation method (**SP**). Note that a combination method may return only matching result of **ML** or only matching result of **SP** or a weighted sum of matching results of both **ML** and **SP**. The experiment methodology is as follows:

- Firstly, the element matcher produces a matching result (**ML**).
- Next, the structure matcher take ML as input and produces another matching

result (**SP**).

- Three automatic weighted sum methods discussed above combine ML and SP to produce combination results such as **HW**, **LC** and **DWS** respectively.
- A filter threshold value $\theta$ is used to select the final mappings for SP, HW and LC. Note that, the similarity score of mappings in ML is 1.0, therefore, changing the filter threshold value does not impact to its matching quality. Besides, our method DWS automatically determines the filter threshold value for each matching scenario, so changing the filter threshold value does not impact to its matching quality either.
- Finally, the H-mean Fmeasure value for each of five matching results ML, SP, HW, LC and DWS is computed.

Fig.5.7 shows the H-mean Fmeasure value for each of five combination methods i.e., ML, SP, HW, LC and DWS overall 21 real test cases of Conference dataset. We chose this dataset because the ontologies in these test cases are highly different in terms of both terminology and structure, which makes sure that using element matcher or structure matcher solely is inadequate.

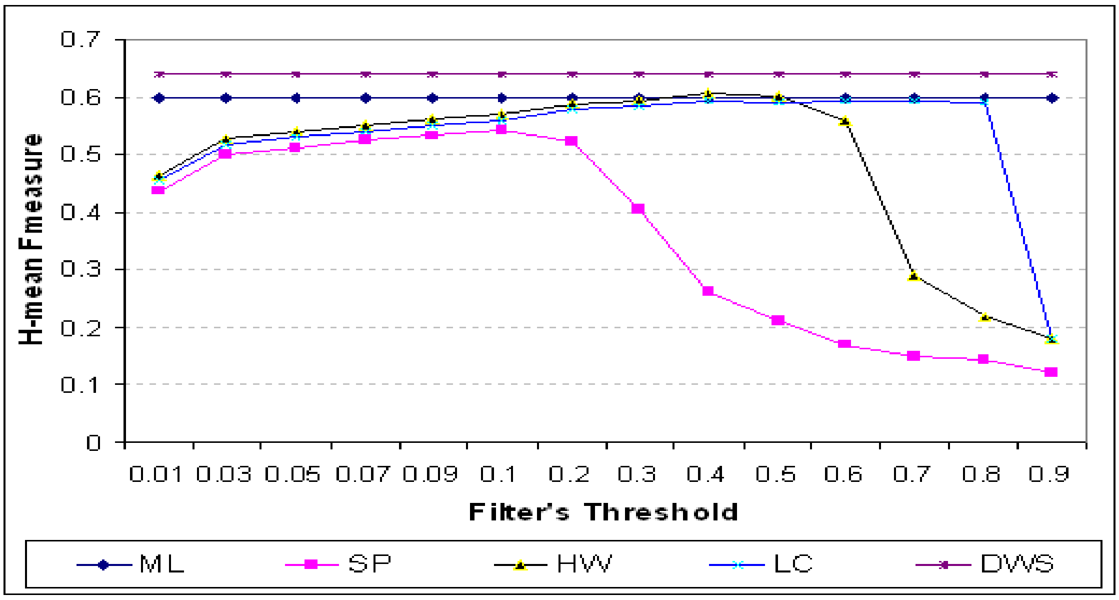


Figure 5.7: Comparison of different methods to combine results of element and structure matchers

The first observation from Fig.5.7 is that the matching quality of SP is lower than that of ML. Here, the H-mean Fmeasure of ML is 0.604, whereas, the maximum H-mean Fmeasure of SP is 0.541 when $\theta = 0.1$. In fact, SP discovered many additional mappings (e.g., (cmt.owl#Paper ≡ confOf.owl#Contribution), etc.) that ML did not. However, SP also discovered many more incorrect mappings when the threshold value is low. It is important to show us three remarks. Firstly, despite the fact that the structural features are more important than the terminological features to describe an entity in an ontology, exploiting the latter ones seems to be more effective than exploiting the former in discovering mappings. Secondly, in the real ontology matching scenarios, matching result relying only on structure matcher like in [67, 98] is not sufficient or do not provide good quality of matching. Finally, in order to overcome the weakness of SP, we need not only an appropriate combination method but also an appropriate filter to select a high quality of final mappings.

We also observe that among three automatic combination methods discussed above, our method DWS outperforms the others. Here, the H-mean Fmeasure of DWS is 0.638, whereas, the maximum H-mean Fmeasure of HW is 0.606 and the maximum H-mean Fmeasure of LC is 0.595. In fact, thanks to the dynamic setting of the weights and the filter's threshold value, DWS improves the matching result of ML by adding new correct mappings obtained by SP. For example, in the matching scenario conOf.owl vs. edas.owl, ML discovers (Event, ≡, ConferenceEvent, 0.0) and (write, ≡, hasRelatedPaper, 0.0), SP discover (Event, ≡, ConferenceEvent, 0.16) and (write, ≡, hasRelatedPaper, 0.18). Three methods DWS, HW and LC produce the same results (Event, ≡, ConferenceEvent, 0.16) and (write, ≡, hasRelatedPaper, 0.18). Besides, DWS automatically determines the filter's threshold for this matching scenario (i.e., $\theta = 0.14$). Therefore, these two mappings passed the filter are selected in final mapping result. Whereas, in the cases of HW and LC, if the threshold value $\theta$ is greater than 0.2, these mappings will be ignored; if the threshold value $\theta$ is smaller 0.1, many other inccorect mappings will pass the filter, consequently, the matching quality is decreased.

Finally, we conclude from this experiment that our dynamic weighted sum method fulfills the two following requirements: (i) automatic setting weights for each matcher and, (ii) automatic setting filter's threshold for the mapping selection process. Moreover, the experimental result shows that our method outperforms the other considered methods.

## 5.5 Conclusions

In this chapter, we have presented our approach to deal with the problem of matcher combination. In particular, we have proposed a Machine Learning based method to combine different terminological similarity measures. We have also proposed a

Dynamic Weighted Sum method to combine matching results of a terminological matcher and a structural matcher.

In terms of the Machine Learning based Combination method, we have transformed the problem of discovering mappings to a problem of binary classification of pairs of entities into predefined classes. In particular, if two entities are classified to a given class with a value 1.0, they are matched; otherwise, they are not. On the other hand, we have proposed a strategy to select similarity measures to combine. This strategy is based on the classification of terminological similarity measures that we have described in Chapter 3.

In terms of the Dynamic Weighted Sum method, we have proposed an algorithm to automatically estimate the weight values for both terminological and structural matchers. Moreover, this method is able to automatically determine the threshold value used for selecting combined matching results in the given matching scenario.

The experiments in Section 5.4 show that both of our methods outperform the two existing automatic combination methods (Harmonic Adaptive Weight Sum and Local Confidence Weighted Sum). Regarding the Machine Learning based combination method; we have concluded that it works best with a Decision Tree - J48 model.

# Chapter 6

# Mapping Selection

In an ontology matching system, mapping selection is an important task because it is involved in many matching phases. In particular, it can be used as a filter to select the high possible candidate mappings before performing the main matching process. During the matching process, it can be used as an internal filter for individual matchers. Indeed, a simple individual matcher is a combination of a similarity measure and a mapping filter. The filter selects the best candidate mappings with respect to the similarity values computed by the similarity measure. The aim of this processing is to reduce the noise data that may be passed as input to another individual matcher. Finally, mapping selection is used at the end of the matching process in order to produce a final result to the user. It becomes more important for the user because it save a lot of post-match effort to verifying the correctness of mapping result.

Mapping selection is to select a subset of mappings from all possible mappings of entities of the two ontologies being matched, that satisfies some given criteria. Each criteria leads to a strategy to select the most appropriate mappings. However, they share a common aim, which is to eliminate suspicious incorrect and inconsistent mappings.

Basically, three following criteria, i.e., 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. The first two filters, i.e., Threshold Filter and Cardinality Filter are ontology independent extraction methods. Their functions do not use any semantic information encoded in the input ontologies. They are will be presented in section 6.1.

Whereas, the Semantic Filter is an ontology dependent extraction method. It takes additional semantic information of entities in the input ontologies in consideration to select the best mappings. The semantic filter will be presented in the next section 6.2. In this section, our main contribution lies on the *Fast Semantic Filtering* method, which effectively and efficiently refines the discovered mappings in the large scale ontology matching.

In the section 6.3, we present the evaluation of the performance of our method dealing with large scale ontology matching. In addition, we compare the performance of our method and method provided by ALCOMO tool [66].

## 6.1 Non Semantic Selection Methods

In this section, we discuss Threshold Filter and Cardinality Filter, which are based on the order of similarity values and the constraint on the matching cardinality.

### 6.1.1 Threshold Filter

A Threshold Filter is a simple filter that selects mappings by comparing their similarity values with a predefined threshold value. Indeed, after performing similarity computation by an individual or a combination of several similarity measures, similarity values are assigned to each candidate mapping. Because the similarity value reflects the degree of confidence that two entities are similar, the higher the similarity value is, the more likely two entities are matched. Therefore, we can use a Threshold Filter to select only pairs of entities, whose similarity values are higher than some predefined threshold value. However, this type of filter may produce multi mapping result, where one entity matches with several other entities. Fig. 6.1 shows an example of using a Threshold Filter.

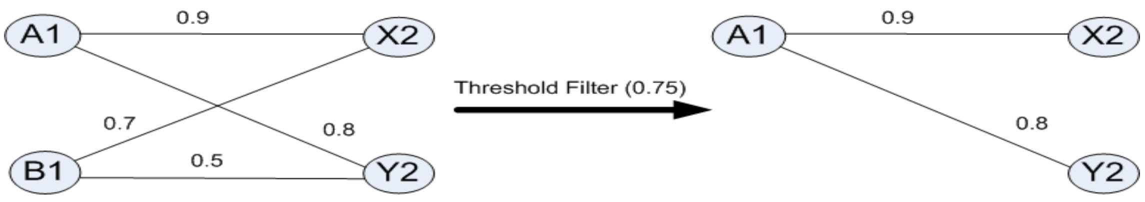


Figure 6.1: An example of a threshold filter

### 6.1.2 Cardinality Filter

Matching cardinality is a constraint to limit multi mapping, where one or several entities of one ontologies may match to one or several entities of the other ontologies.

The aim of the Cardinality Filter is to comply the matching cardinality rigorously to eliminate the less similar mappings from the multi mappings.

Among several types of matching cardinality, one-to-one (1:1) mapping is widely used in the ontologies matching field. In fact, when domain knowledge is conceptualized in an ontology, the ontology developers always try to avoid or minimize number of duplicate entities. Therefore, when we run matching between entities of two distinct ontologies, we mainly expect `1:1` cardinality matching, which means one entity in the source ontology is matched to maximum one entity in the target ontology. Of course, there may be other type of cardinality such as `1:n`, `m:1` or `m:n`, but they do not frequently happen.

In the ontology matching field, **Greedy Filter** and **Maximum Assignment** are two main methods usually used to extract mappings with 1:1 matching cardinality. Here, we first present their algorithms, then, we will discuss the way to extend them to deal with multi mapping.

### Greedy Filter

The main steps of this filter are described in the Algorithm.4. Here, *SortDescending* function sorts the input set of mappings in descending order of their confidence value. In each iteration, the mapping with the highest confidence value is extracted. Then, this mapping is added to the result set if the confidence value (*getScore* function) is higher than threshold $\theta$, otherwise the `while` loop will stop. Function $GetRelated(M, m)$ is defined to return all other mappings in $M$, whose source or target entities are the same with ones in $m$.

An example of using a Greedy Filter with different threshold can be seen in Fig.6.2

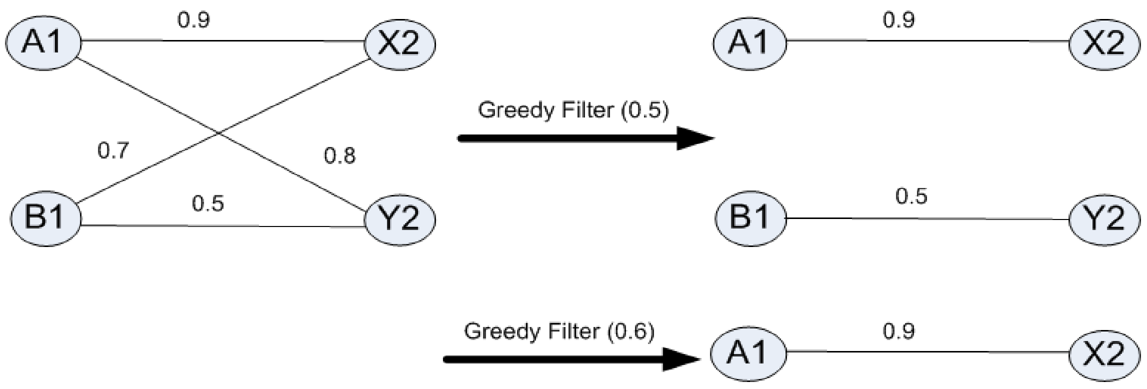


Figure 6.2: An example of using greedy filter

In order to extend this Filter for obtaining multi-mapping (1:n, m:1 or n:m), we

**Algorithm 4:** Greedy Selection with threshold $\theta$

```
input  : $M_{orig}$ = {($e_i$,$e_j$,$\equiv$,c), $c \in [0..1]$}
         $\theta \in (0..1]$ : threshold value
output: $M_{sub}$ = {($e_p$,$e_q$,$\equiv$,c), $c \in [0..1]$}
1  $M_{sub} \leftarrow \emptyset$
2  SortDescending($M_{orig}$)
3  while $M_{orig} \neq \emptyset$ do
4      $m \leftarrow$ RemoveFirstElement($M_{orig}$)
5      if getScore($m$) $\geq \theta$ then
6          $M_{sub} \leftarrow M_{sub} \cup \{m\}$
7      else
8          return $M_{sub}$
9      end
10     for $m' \in$ GetRelated($M_{orig}, m$) do
11         RemoveElement($M_{orig}, m'$)
12     end
13 end
14 return $M_{sub}$
```

propose a simple technique as follows. Assume that we would select the maximum top-K entities from one ontology that match to one entity of another ontology. Thus, in each of K iterations, the following steps will be executed:

- Step 1: Run the Greedy Filter overall the candidate mappings to obtain a subset of them as the best mappings.
- Step 2: Copy the returned result to another place and the similarity value for each of those mappings in the original collection of candidates is set to 0. Repeat the Step 1.

Fig.6.3 shows an example of using a Greedy Filter with 2 iterations.

One of the advantage of this Filter is that is run very efficient. It complexity is $O(N)$, where N is the total number of candidate mappings.

### Maximum Assignment

Contrary to the Greedy Filter, which is based on sequences of local decisions, the Maximum Assignment Filter finds a solution that is optimal from a global point of view. In particular, it selects a subset of 1:1 mappings that maximizes the total similarity values.

For solving an optimal solution, Hungarian assignment algorithm is usually implemented. The main steps of the Hungarian algorithm are shown as follows.

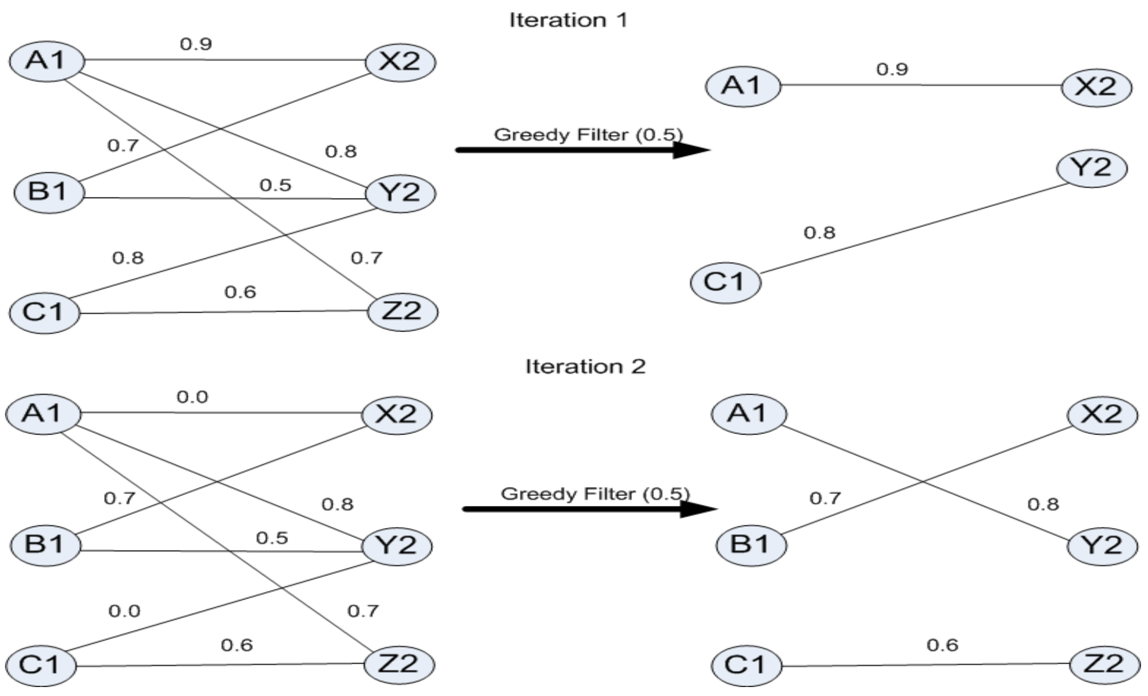


Figure 6.3: An example of using greedy filter with 1:2 matching cardinality

- **Step 0**: Create an $NxM$ matrix called the cost matrix in which each element represents the distance value (i.e., 1 - similarity value) of assigning one of $N$ entities of the source ontology to one of m entities of the target ontology. Rotate the matrix so that there are at least as many columns as rows and let $K = min(N, M)$.

- **Step 1**: For each row of the matrix, find the smallest element and subtract it from every element in its row. Go to Step 2.

- **Step 2**: Find a zero (Z) in the resulting matrix. If there is no starred zero in its row or column, star Z. Repeat for each element in the matrix. Go to Step 3.

- **Step 3**: Cover each column containing a starred zero. If K columns are covered, the starred zeros describe a complete set of unique assignments. In this case, Go to DONE, otherwise, Go to Step 4.

- **Step 4**: Find a non covered zero and prime it. If there is no starred zero in the row containing this primed zero, Go to Step 5. Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value and Go to Step 6.

- **Step 5**: Construct a series of alternating primed and starred zeros as follows. Let $Z_0$ be the uncovered primed zero found in Step 4. Let $Z_1$ denote the starred zero in the column of $Z_0$ (if any). Let $Z_2$ denote the primed zero in the row of $Z_1$ (there will always be one). Continue until the series terminate at a primed zero that has no starred zero in its column. Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix. Return to Step 3.

- **Step 6**: Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any stars, primes, or covered lines.

An example of using a Maximum Assignment Filter can be seen in Fig.6.4

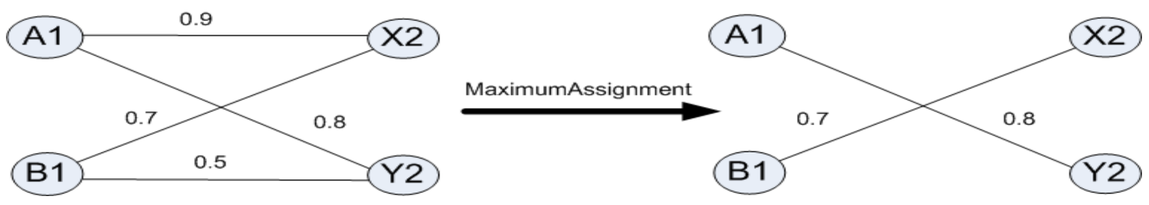


Figure 6.4: An example of using maximum assignment filter

In order to extend this Filter for obtaining multi mappings, we can run this method several times. Similar to the technique used for Greedy Filter, the selected mappings at the end of each time of running this method are stored before being set 0 for their value in the next time. However, this filter works very slow with $O(N^3)$ complexity.

## 6.2 Semantic Selection Methods

The aim of the Semantic Filter is to detect and reject inconsistent mappings by exploring semantic information of entities in the input ontologies. By applying this filter after computing similarity values for all candidate mappings, we can obtain a consistent alignment, which contains the best mappings between the input ontologies. In this section, we first present the use of Description Logic in ontology. Then, the notion of inconsistent and unstable mappings will be defined. Next, we discuss methods to detect and eliminate them in order to obtain the optimal consistent set of mappings.

### 6.2.1 Description Logic and Ontology

First of all, we are going to introduce the fundamental of Description Logic (DL) underlying ontology. In particular, we will focus the Web Ontology Language (OWL), which is a knowledge representation language standardized by the World Wide Web Consortium (W3C). Basically, the main functions of OWL (especially OWL-DL) are indeed very similar to those of DLs, which prodive means to model the relationships between entities in a domain of interest. It is therefore not surprising that description logics have had a major influence on the development of OWL and the expressive features that it provides.

Like DL, OWL ontology describes all logic expressions upon a collection of vocabulary called a signature. According to the specification of OWL language, we adopt the following definition of a signature in [66]:

**Definition 4.** *(Signature). A signature $S$ is a quadruple $\boldsymbol{S} = \langle \boldsymbol{C}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{I} \rangle$ where $\boldsymbol{C}$ is a set of concept names, $\boldsymbol{P}$ is a set of object property names, $\boldsymbol{R}$ is a set of data property names, and $\boldsymbol{I}$ is a set of individual names. The union $\boldsymbol{P} \cup \boldsymbol{R}$ is referred to as the set of property names.*

Unlike a database, a DL ontology does not fully describe a particular situation or "*state of the world*"; rather it consists of a set of statements, called axioms or assertions, each of them must be true in the described situation [53]. Therefore, an ontology usually consists of both terminological axioms $TBox$ and assertions $ABox$. Here, the terminological axioms $TBox$ are used to define the meaning of a named concept or property by clarifying its relations to the other concepts in the ontology. Opposed to an axiom, an assertion ($ABox$) is used to make a statement about an instance by describing its qualities in terms of concept membership and relations to other instances. Thus, a definition of an ontology from the point of view of DL is as follows:

**Definition 5.** *(Ontology). Given an ABox $\boldsymbol{A}$ and a TBox $\boldsymbol{T}$ in $\boldsymbol{S} = \langle \boldsymbol{C}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{I} \rangle$. The union $O = A \cup T$ is called an ontology in $\boldsymbol{S}$. $\boldsymbol{S}$ is called the signature of $\boldsymbol{O}$ if there exists no $\boldsymbol{S'} = \langle \boldsymbol{C'}, \boldsymbol{P'}, \boldsymbol{R'}, \boldsymbol{I'} \rangle$ such that (i) $\boldsymbol{O}$ is in $\boldsymbol{S'}$ and (ii) $C' \subset C$ or $P' \in P$ or $R' \in R$ or $I' \in I$.*

As their name suggests it, DLs are logics and as such they are equipped with a formal semantics: a precise specification of the meaning of DL ontologies. This formal semantics allows humans and computer systems to exchange DL ontologies without ambiguity, and also makes it possible to use logical deduction to infer additional information from the facts stated explicitly in an ontology - an important feature that distinguishes DLs from other modeling languages such as UML [53]. The computation of inferences is called *reasoning* - one of the most important goal of DL language.

In order to understand the semantic meaning of a complex expression in DL, an interpretation is needed. The general principle is known as the principle of compositionality, where the meaning of a complex expression is determined by the meaning of its constituent expressions and the rules used to combine them. The definition of a DL interpretation is the following:

**Definition 6.** *(Interpretation).*

*Given an ontology and its signature* $\boldsymbol{S} = \langle \boldsymbol{C}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{I} \rangle$ *and a datatype theory* $\boldsymbol{D}$*. An interpretation* $\mathcal{I} = \left\langle \Delta^{\mathcal{I}}, \Delta^{\mathcal{I}}_{D}, \cdot^{\mathcal{I}} \right\rangle$ *consists of a set* $\Delta^{\mathcal{I}}$*, which is the abstract domain; a set* $\Delta^{\mathcal{I}}_{D}$*, which is the concrete domain (concrete data values); and a function* $\cdot^{\mathcal{I}}$ *that maps every concept name in* $\boldsymbol{C}$ *to a subset of* $\Delta^{\mathcal{I}}_{D}$*, every object property name in* $\boldsymbol{P}$ *to a subset of* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*, every data property name in* $\boldsymbol{R}$ *to a subset of* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}_{D}$*, every individual name in* $\boldsymbol{I}$ *to an element of* $\Delta^{\mathcal{I}}$*, every datatype in* $\boldsymbol{D}$ *to a subset of* $\Delta^{\mathcal{I}}_{D}$*, and every data constant to a value in* $\Delta^{\mathcal{I}}_{D}$*.*

*Furthermore,*

$$\begin{array}{rcl}
\top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \quad (owl:Top) \\
\bot^{\mathcal{I}} & = & \emptyset \quad (owl:Bottom) \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \quad (owl:complementOf) \\
(B \sqcap C)^{\mathcal{I}} & = & B^{\mathcal{I}} \cap C^{\mathcal{I}} \quad (owl:intersectionOf) \\
(B \sqcup C)^{\mathcal{I}} & = & B^{\mathcal{I}} \cup C^{\mathcal{I}} \quad (owl:unionOf) \\
\{o_1, \ldots, o_n\}^{\mathcal{I}} & = & \{o_1^{\mathcal{I}}, \ldots, o_n^{\mathcal{I}}\} \quad (enumeration) \\
(\exists P.C)^{\mathcal{I}} & = & \{x \mid \exists y \, \langle x, y \rangle \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \quad (owl:someValuesFrom) \\
(\forall P.C)^{\mathcal{I}} & = & \{x \mid \forall y \, \langle x, y \rangle \in P^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\} \quad (owl:allValuesFrom) \\
(\exists_{\leq n} P)^{\mathcal{I}} & = & \{x \mid \#\{\langle x, y \rangle \in P^{\mathcal{I}}\} \leq n\} \quad (owl:maxCardinality) \\
(\exists_{\geq n} P)^{\mathcal{I}} & = & \{x \mid \#\{\langle x, y \rangle \in P^{\mathcal{I}}\} \geq n\} \quad (owl:minCardinality) \\
(\exists R.D)^{\mathcal{I}} & = & \{x \mid \exists y \, \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \quad (owl:someValuesFrom) \\
(\forall R.D)^{\mathcal{I}} & = & \{x \mid \forall y \, \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in D^{\mathcal{I}}\} \quad (owl:allValuesFrom) \\
(\exists_{\leq n} R)^{\mathcal{I}} & = & \{x \mid \#\{\langle x, y \rangle \in R^{\mathcal{I}}\} \leq n\} \quad (owl:maxCardinality) \\
(\exists_{\geq n} R)^{\mathcal{I}} & = & \{x \mid \#\{\langle x, y \rangle \in R^{\mathcal{I}}\} \geq n\} \quad (owl:minCardinality)
\end{array}$$

*where* $B$ *and* $C$ *are concept descriptions in* $\boldsymbol{S}$*,* $D$ *is a datatype defined in* $\boldsymbol{D}$*,* $P$ *is* *an object property description in **S**, R is a data property description in **S**,* $n \in R+$*, and* $o_1, \ldots, o_n \in \boldsymbol{I}$ *are individual names. [66]*

Through the DL interpretation, the satisfiability of a DL statement (i.e., axiom, assertion) can be defined as follows:

**Definition 7.** *(Satisfiability).*

*Given interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \Delta^{\mathcal{I}}_{D}, \cdot^{\mathcal{I}} \rangle$*.* $\mathcal{I}$ *satisfies an axiom:*

$$\begin{aligned}
C_1 \sqsubseteq C_2 \quad & iff \quad C_1^{\mathcal{I}} \sqsubseteq C_2^{\mathcal{I}} \qquad (conceptinclusion)\\
P_1 \sqsubseteq P_2 \quad & iff \quad P_1^{\mathcal{I}} \sqsubseteq P_2^{\mathcal{I}} \qquad (objectpropertyinclusion)\\
R_1 \sqsubseteq R_2 \quad & iff \quad R_1^{\mathcal{I}} \sqsubseteq R_2^{\mathcal{I}} \qquad (datapropertyinclusion)\\
trans(P) \quad & iff \quad \langle x, y \rangle \in R^{\mathcal{I}} \wedge \langle y, z \rangle \in R^{\mathcal{I}} \rightarrow \langle x, z \rangle \in R^{\mathcal{I}} (objectpropertytransitivity)
\end{aligned}$$

*where* $C_1$ *and* $C_2$ *are concept descriptions,* $P_1$ *and* $P_2$ *are object property descriptions, and* $R_1$ *and* $R_2$ *are data properties. I satisfies an equivalence axiom* $X \equiv Y$ *iff* $\mathcal{I}$ *satisfies* $X \sqsubseteq Y$ *and* $\mathcal{I}$ *satisfies* $Y \sqsubseteq X$*. Furthermore, I satisfies an assertion:*

$$\begin{aligned}
C(a) \quad & iff \quad a^{\mathcal{I}} \in C^{\mathcal{I}} \qquad (conceptassertion)\\
P(a, b) \quad & iff \quad \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \qquad (objectpropertyassertion)\\
P(a, d) \quad & iff \quad \langle a^{\mathcal{I}}, d^{\mathcal{I}} \rangle \in R^{\mathcal{I}} \qquad (datapropertyassertion)\\
a = b \quad & iff \quad a^{\mathcal{I}} = b^{\mathcal{I}} \qquad (equality)\\
a \neq b \quad & iff \quad a^{\mathcal{I}} \neq b^{\mathcal{I}} \qquad (inequality)
\end{aligned}$$

*where* $C$ *is a concept description,* $P$ *is an object property description,* $R$ *is a data property,* $a$ *and* $b$ *are individuals, and* $d$ *is a data value. [66]*

Due to this definition, an ontology divides the set of interpretations into those interpretations that do not satisfy the ontology and those interpretations that satisfy the ontology. The latter ones are called models of the ontology.

**Definition 8.** *(Model).*

*An interpretation* $\mathcal{I}$ *is a model for an ontology **O**, iff* $\mathcal{I}$ *satisfies each axiom and each assertion in **O**. [66]*

Thus a model is an abstraction of a state of the world that satisfies all axioms in the ontology. An ontology is *consistent* if it has at least one model. A DL statement $a$ is a consequence of an ontology $O$ (or $O$ entails $a$) if $a$ holds in every model of $O$.

**Definition 9.** *(Entailment).*

*An ontology $\boldsymbol{O}$ entails an assertion or axiom a, iff each model for $\boldsymbol{O}$ is also a model for a. An ontology $\boldsymbol{O}$ entails a set of assertions or axioms A, iff each model for $\boldsymbol{O}$ is also a model for each $a \in A$. We write $O \models a$ if $\boldsymbol{O}$ entails a; if $\boldsymbol{O}$ does not entail a we write $O \not\models a$. [66]*

Now, we can define the notion of concept and property (un)satisfiability as well as the notion of ontology (in)coherence. A concept $C$ is defined to be unsatisfiable iff each model $\mathcal{I}$ of $O$ maps $C$ to the empty set, i.e., an instance of $C$ cannot exist for logical reasons.

**Definition 10.** *(Concept/Property Unsatisfiability).*

*Given an ontology $\boldsymbol{O}$ and its signature $\boldsymbol{S}$. A concept description $C$ (property description $P$) in $\boldsymbol{S}$ is unsatisfiable in $\boldsymbol{O}$ iff for each model $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \Delta_D^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ of $\boldsymbol{O}$ we have $C^{\mathcal{I}} = \emptyset$; ($P^{\mathcal{I}} = \emptyset$). [66]*

**Definition 11.** *(Incoherence).*

*Given an ontology $\boldsymbol{O}$ and its signature $\boldsymbol{S} = \langle \boldsymbol{C}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{I} \rangle$. $\boldsymbol{O}$ is incoherent iff there exists an unsatisfiable $C \in \boldsymbol{C}$, $P \in \boldsymbol{P}$ or $R \in \boldsymbol{R}$. Otherwise $\boldsymbol{O}$ is called coherent. [66]*

As we mentioned above, the task of a DL reasoning system is to infer additional knowledge from a explicitly stated collection of axioms and assertions. Therefore, we can make use of a DL reasoner (e.g., Pellet, Hermit, Fact++, etc.) to verify the entailment of an axiom/assertion, the satisfiability of a concept/property or the coherent of an ontology.

### 6.2.2 Inconsistent Mappings

From the previous section, we understand what is an unsatisfied concept/property and what is an incoherent ontology. Based on those ideas, now we are going to define the notion of inconsistent mappings between ontologies to be matched.

Assume that $A$ is an alignment (i.e., a collection of mappings) discovered from two ontologies $O_1$ and $O_2$. Thanks to $A$, the two ontologies $O_1$ and $O_2$ can be merged into one ontology by the following rule.

**Definition 12.** *(Merged ontology).*

*The merged ontology $O_1 \cup_A O_2$ of two ontologies $O_1$ and $O_2$ connected via alignment $A$ is defined as:*

$$O_1 \cup_A O_2 = O_1 \cup O_2 \cup \{t(m) \mid m \in A\}$$

*where, $t$ is a function contering each mapping $m = \langle O_1 : e_1, O_2 : e_2, \equiv, c\rangle$ into an equivalent axiom of $O_1 \cup_A O_2$. [66]*

By adding the mappings of $A$ as bridges between two ontologies, we have a unique structural hierarchy connecting their all entities. Hence, we can make use of a DL reasoning system on the merged ontology in order to discover new information. However, those bridges may make the merged ontology become incoherent, where several concepts or properties of the input ontologies become unsatisfied. In that case, $A$ is **incoherent** with respect to the input ontologies. Therefore, some mappings in the alignment $A$ are inconsistent.

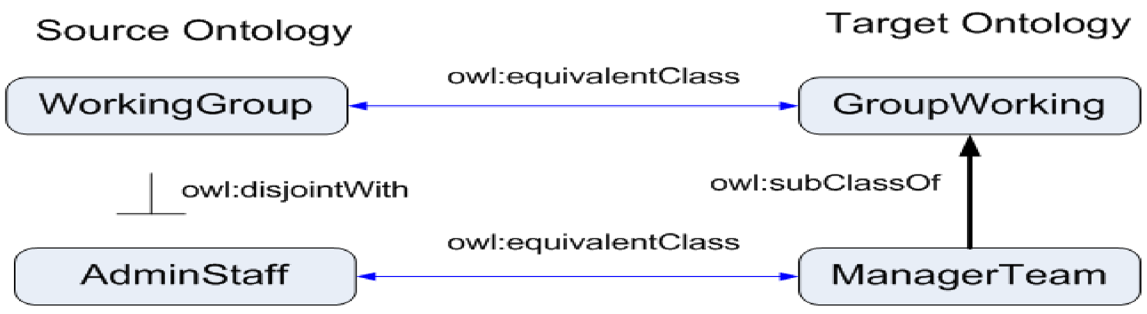


Figure 6.5: An example of inconsistent mappings

We present an example of incoherent mappings in Fig.6.5. If two mappings $\langle WorkingGroup, GroupWorking\rangle$ and $\langle AdminStaff, ManagerTeam\rangle$ are consistent, then in the merged ontology we can infer the following information:

$$\begin{aligned}
WorkingGroup &\perp AdminStaff \wedge WorkingGroup \equiv GroupWorking \\
&\rightarrow AdminStaff \perp GroupWorking \\
AdminStaff &\equiv ManagerTeam \wedge AdminStaff \perp GroupWorking \\
&\rightarrow ManagerTeam \perp GroupWorking \\
ManagerTeam &\sqsubseteq GroupWorking \wedge ManagerTeam \perp GroupWorking \\
&\rightarrow ManagerTeam \sqsubseteq \perp
\end{aligned}$$

In that case, the concept **ManagerTeam** is not satisfied the merged ontology. Therefore, the two mappings are inconsistent.

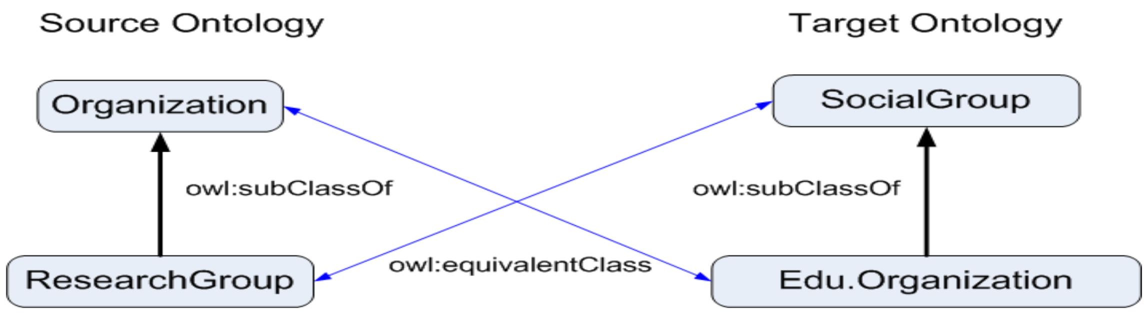


Figure 6.6: An example of unstable mappings

Let us present another example in Fig.6.6. If two mappings $\langle Organization, Edu.Organization\rangle$ and $\langle ResearchGroup, SocialGroup\rangle$ are consistent, then in the merged ontology we can infer the following information:

$$
\begin{aligned}
ResearchGroup &\sqsubseteq Organization \wedge Organization \equiv Edu.Organization \\
&\rightarrow ResearchGroup \sqsubseteq Edu.Organization \\
ResearchGroup &\equiv SocialGroup \wedge ResearchGroup \sqsubseteq Edu.Organization \\
&\rightarrow SocialGroup \sqsubseteq Edu.Organization \\
SocialGroup &\sqsubseteq Edu.Organization \wedge Edu.Organization \sqsubseteq SocialGroup \\
&\rightarrow Edu.Organization \equiv SocialGroup \\
Organization &\equiv Edu.Organization \wedge ResearchGroup \sqsubseteq Edu.Organization \\
&\rightarrow Organization \sqsubseteq ResearchGroup \\
ResearchGroup &\sqsubseteq Organization \wedge Organization \sqsubseteq ResearchGroup \\
&\rightarrow ResearchGroup \equiv Organization
\end{aligned}
$$

In that case, we can infer two equivalences in the two input ontologies. It is possible because if concept $A$ is equivalent to concept $B$ then concept $A$ may be considered as a subsumption of concept $B$. However, if there is any other constraint in the input ontologies stating that those concepts cannot be equivalent then the two mappings will become inconsistent. In [66], those mappings are called unstable mappings.

### 6.2.3 Review Existing Methods

So far we understand the fundamental of Description Logic underlying ontology and notion of the inconsistent mappings. In order to extract the best mappings from the discovered ones, now we have to detect the set of inconsistent mappings and select the correct subset of mappings with minimum incoherence. In the following, we review some approaches that exploit the semantic underlying ontologies in order to avoid logical problems in the produced mappings. In particular, we will discuss a semantic verification method used in ASMOV [42] and (in)complete reasoning method used in ALCOMO and Logmap.

#### Semantic Verification with ASMOV

ASMOV, which stands for Automated Semantic Mapping of Ontologies with Verification [42], is one of the top three alignment tools at OAEI campaigns (2008, 2009 and 2010). The main idea of ASMOV is not to find semantically invalid or unsatisfiable alignments, but rather to remove correspondences that are less likely to be satisfiable based on the information present in the ontologies.

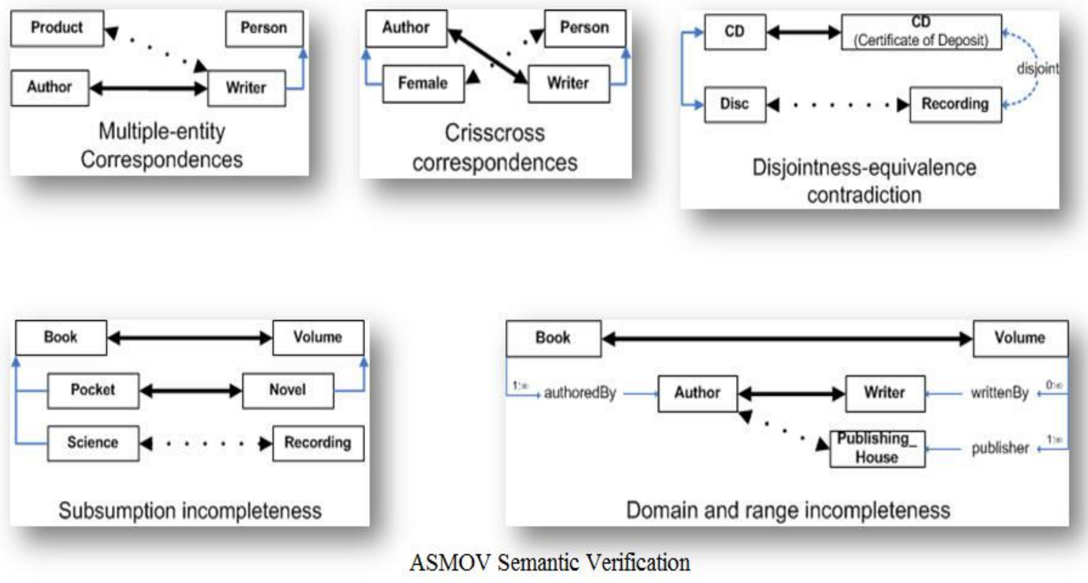

Product
Person
Author
Writer
Multiple-entity Correspondences
Author
Person
Female
Writer
Crisscross correspondences
CD
CD (Certificate of Deposit)
disjoint
Disc
Recording
Disjointness-equivalence contradiction
Book
Volume
Pocket
Novel
Science
Recording
Subsumption incompleteness
Book
Volume
1:∞
authoredBy
Author
Writer
writtenBy
0:∞
Publishing_House
publisher
1:∞
Domain and range incompleteness
ASMOV Semantic Verification

Figure 6.7: Inconsistent mapping patterns used in ASMOV [42]

According to the system description, ASMOV uses a list of patterns to detect pairs of conflict correspondences (i.e., mappings). Fig. 6.7 shows examples of the inconsistent mapping patterns used in ASMOV. In particular, they are including the following patterns:

- Multiple-entity correspondences. This pattern occurs when one entity in one ontology is matched to some different entities in another ontology, whereas, the matching cardinality is 1:1.

- Criss cross correspondences. Two correspondences $\langle a_1, a_2 \rangle$ and $\langle b_1, b_2 \rangle$ are found by this pattern if $O_1 \models a_1 \sqsubseteq a_2$ and $O_2 \models b_2 \sqsubseteq b_1$.

- Disjointness subsumption contradiction. Two correspondences $\langle a_1, a_2 \rangle$ and $\langle b_1, b_2 \rangle$ are an instance of this pattern if $O_1 \models a_1 \sqsubseteq a_2$ and $O_2 \models b_1 \sqsubseteq \neg b_2$ or $O_1 \models a_1 \sqsubseteq \neg a_2$ and $O_2 \models b_1 \sqsubseteq b_2$.

- Subsumption and equivalence incompleteness. Two correspondences $\langle a_1, a_2 \rangle$ and $\langle b_1, b_2 \rangle$ are found by this pattern if $O_1 \models a_1 \sqsubseteq a_2$ (or $O_1 \models a_1 \equiv a_2$) but $O_2 \not\models b_1 \sqsubseteq b_2$ (or $O_2 \not\models b_1 \equiv b_2$).

- Domain and range incompleteness. A concept-concept mapping $\langle c_1, c_2 \rangle$ and a property-property mapping $\langle p_1, p_2 \rangle$ are detected by this pattern if $O_1 \models c_1 \sqsubseteq \mathit{domain}(p_1)$ and $O_1 \not\models c_2 \sqsubseteq \mathit{domain}(p_2)$; or $O_1 \models c_1 \sqsubseteq \mathit{range}(p_1)$ and $O_1 \not\models c_2 \sqsubseteq \mathit{range}(p_2)$

In each iteration, ASMOV extracts a set of conflicts from the pre-alignment which results from the similarity computation. In order to remove inconsistent correspondences, ASMOV makes use of a greedy method. That is the correspondence with the lowest confidence value is eliminated from the set of conflicts. The removed correspondences are then stored in a list of removals and they will be not touched in the next iteration.

Obviously, the detection of conflict correspondences is very important in the ASMOV algorithm. It is because if a correct correspondence was removed in an iteration, it will be never in the final alignment. Therefore, the detecting patterns should be highly precise. Our observations of these patterns are the following:

- Firstly, the *multiple-entity correspondences* only works in a specific circumstance when the matching quality is 1:1. In case the restriction on matching cardinality is different to 1:1, the first pattern may reject many incorrect mappings.

- As it was showed in the examples in section 6.2.2, the *disjointness subsumption contradiction* is a strong conflict pattern because it makes the merged ontology incoherent. One of them is incorrect and we have to remove it.

- The degree of conflict of the *criss cross correspondences* and the *subsumption and equivalence incompleteness* patterns are weaker than that of the *disjointness subsumption contradiction* pattern. They are only used in case of the input ontologies are complete. When the ontologies are incomplete, they become uncertain. For example, the *criss cross correspondences* pattern leads to entities in the mappings become equivalent, i.e., $O_1 \models a_1 \equiv a_2$, which means $O_1 \models a_1 \sqsubseteq a_2 \wedge a_2 \sqsubseteq a_1$; and $O_2 \models b_1 \equiv b_2$, which means $O_2 \models b_1 \sqsubseteq b_2 \wedge b_2 \sqsubseteq b_1$. In this case, we may consider that $O_1 \models a_2 \sqsubseteq a_1$ and $O_2 \models b_1 \sqsubseteq b_2$ are two missing subsumption relations in the two ontologies. It is possible because ontologies in general are incomplete.

- Finally, the *domain and range incompleteness* pattern is weak and maybe is not correct. In fact, a property may have different domains and ranges.

Based on the observations above, we realize that the detection and elimination of the *disjointness subsumption contradiction* pattern is necessary, whereas, the other patterns are less certain and they cannot theoretically work for all cases. However, according to the good results of ASMOV in the recent years of OAEI campaign, in practice, we can reuse some of them, for example *criss cross correspondences* pattern which is commonly used in other ontology matching systems [97].

Furthermore, running semantic verification to eliminate inconsistent correspondences at each working iteration is risky. It is because the elimination of a correspondence is based on its current confidence (i.e., similarity) value. It is possible that at some intermediate step, the confidence value of an incorrect correspondence is higher than that of a correct one. Thus, the correct correspondence will be rejected; the incorrect will be remained and it maybe propagates other errors.

**Filtering Method with ALCOMO**

In work [66], the author presents ALCOMO - a state of the art and effective tool for dealing with incoherent alignment. Indeed, ALCOMO transforms this problem into a diagnostic problem. It find a minimal set of mappings $\Delta$ (i.e., alignment diagnosis) from a given set of mappings $A$ between input ontologies $O_1$ and $O_2$ such that $A \backslash \Delta$ is coherent.

The main idea of the ALCOMO is as follows. Firstly, it detects all MIPS (i.e., minimal incoherent preserving sub-TBox) from the given alignment. Here, each MIPS is a minimal conflict set of mappings that make the merged ontology become incoherent. After having a collection of MIPS, ALCOMO applies an optimization method in order to find an optimal alignment diagnosis. By removing mappings of the alignment diagnosis, ALCOMO produces a coherent set of mappings.

Basically, each method provided by ALCOMO can be seen as a combination of two main components such as a reasoning component and a computing optimization component. For each component, several methods have been designed. In particular, ALCOMO supports two types of reasoning such as complete and incomplete methods. It also supports two types of computing optimization such as local optimization alignment diagnosis and global optimization alignment diagnosis.

**Reasoning Component** The aim of the reasoning component is to identify unsatisfied concepts/properties and to check the coherence of a set of mappings. In terms of using a complete reasoning method, ALCOMO makes use of a Description Logic reasoning system (e.g., Pellet, Hermit) to work with the merged ontology. In terms of using an incomplete reasoning method, similar to ASMOV, ALCOMO proposes several conflict patterns in order to detect all MIPS.

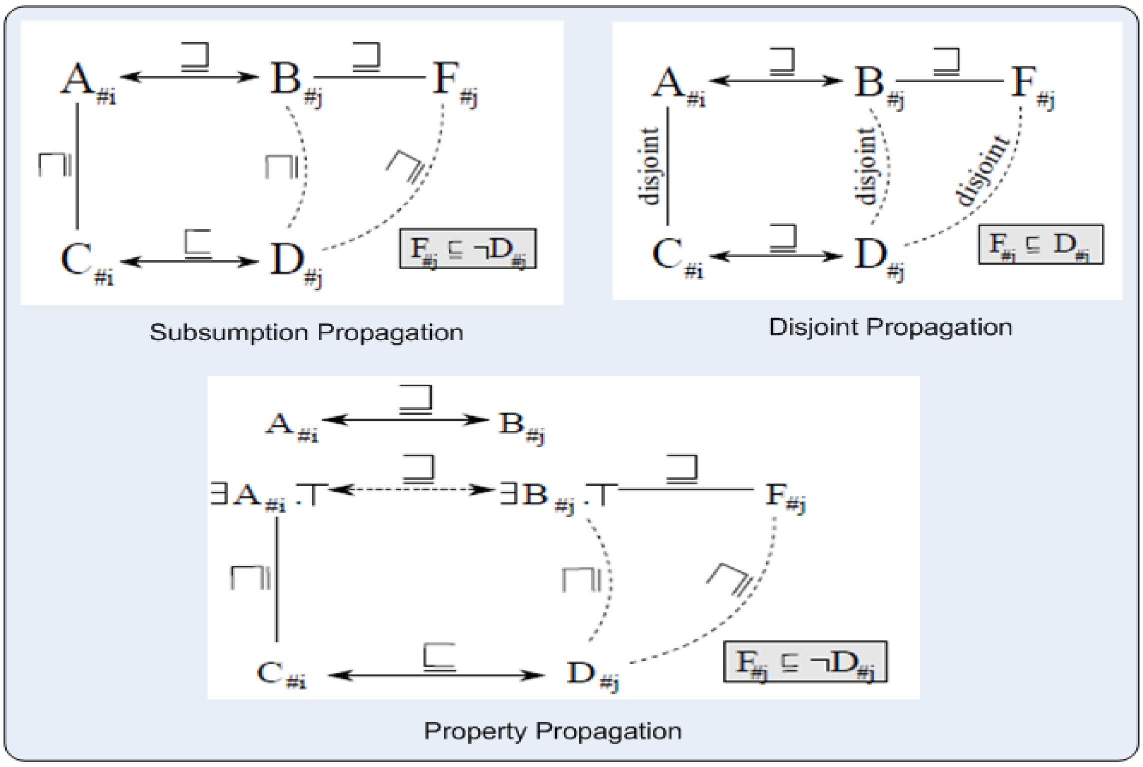


Figure 6.8: The ALCOMO conflict patterns (taken from [66])

According to the system description in [66], ALCOMO uses three patterns shown in the Fig.6.8. Note that, $A_{\#i}$ means entity (i.e., concept, property) $A$ of ontology $O_i$.

- Subsumption propagation. This pattern is quite similar to the *subsumption and equivalence incompleteness* pattern of ASMOV. However, in ASMOV, it is used heuristically with assumption that the input ontologies are complete. In ALCOMO, this pattern is stronger than that of ASMOV due to the fact of disjoint axioms. Indeed, ALCOMO only concludes that two correspondences $\langle A_{\#i}, B_{\#j}\rangle$ and $\langle C_{\#i}, D_{\#j}\rangle$ are in conflict if $O_i \models A_{\#i} \sqsubseteq C_{\#i}$ and there exist a concept $F_{\#j}$ that $O_j \models F_{\#j} \sqsubseteq B_{\#j}$ and $O_j \models F_{\#j} \neg D_{\#j}$. In Fig.6.8, the author drew only the subsumption $\sqsubseteq$ relations. In fact, this pattern can be applied to the equivalent relation ($\equiv$) also.

- Disjointness propagation. This pattern is similar to the previous pattern except that it propagates the disjointness relation instead of subsumption relations along the concept hierarchy of the input ontologies.

- Property propagation. This pattern is similar to the *domain and range incompleteness* pattern of ASMOV but stronger and more explicit. Indeed, a pair of correspondences $M$ including a correspondence between two properties $\langle A_{\#i}, B_{\#j}\rangle$ and a correspondence between two concepts $\langle C_{\#i}, D_{\#j}\rangle$ are in conflict because a reasoning system can infer that $O_i \cup_M O_j \models F_{\#j} \sqsubseteq D_{\#j}$ but $O_j \models F_{\#j} \neg D_{\#j}$.

Obviously, the detection of conflict correspondences in ALCOMO is compeletly derived from logical axioms. There is no assumption or heuristic about the input ontologies like ASMOV has. Indeed, all three patterns depend of the existence of disjointness axioms in the input ontologies. However, in practice, this type of axioms is usually omitted. That is a major drawback that the author of ALCOMO have mentioned in his work [66].

Furthermore, ALCOMO makes use of a reasoning system to perform both complete and incomplete reasoning tasks. This way has both advantages and disadvantages. The major benefit is that the effective inferring algorithms can be reused in order to automatically entail new logical axioms. For example, to check two concepts $C_1$ and $C2$ are disjoint or not, ALCOMO first creates a new axiom such as $OWLDisjointClassesAxiom(C_1, C_2)$; then, ALCOM uses Pellet - a DL reasoner, to verify this axiom. However, there are also two drawbacks of using a reasoner such as time performance and memory usage. Basically, due to the big size of the input ontologies, the classification process of the reasoner on those ontologies or on the merged ontology is time consuming and maybe raise an error exception of out of memory.

**Computing Optimization Diagnosis Component** The aim of this component is to select the minimum diagnosis that if we remove its correspondences we obtain a coherent set of correspondences. In ALCOMO, this task can be done with either local optimal method or glocanl optimal method. The detail of these methods can be read in chapter 6 in [66]. Here, we show only the main ideas of them.

- The main idea of the local optimal method is based on the principle to trust always correspondences with higher confidence in case of a conflict. Therefore, from each MIPS $M$, ALCOMO removes the correspondence with lowest confidence unless another correspondence in $M$ has not yet been removed. At the end, the result is a set of removed correspondences.

- The main idea of the global optimal method is to find a set of coherent correspondences that have the lowest sum of confidences.

According to the experimental analyses of these methods in [66], both methods increase the quality of a given alignment by removing inconsistent correspondences. In addition, the results of applying the global method are slightly better than that of applying the local method. Indeed, the local optimization method is highly sensitive to the order of confidence values, whereas, these kinds of problems are avoided by computing a global optimization, which is determined by an aggregation of confidence values. However, in terms of time performance, the local method outperforms the global method. In fact, given an incoherent alignment $A$, the global method requires $\|A\|$ times to check the unsatisfiability of a specific class, whereas, the local method requires to check $log_2 \|A\|$ times whether there exist some unsatisfiable concepts. Moreover, in the conclusion, the author also stated that when the global optimal diagnosis contains more than 40 correspondences, the global method will raise a problem and it does not return any result. Finally, it is important to note that these methods have been designed for the 1:1 matching cardinality. That is if a matching scenario allows 1:n, m:1 or n:m matching cardinality, these methods may remove correct correspondences.

### 6.2.4 Fast Semantic Filtering Method

In order to reduce the weaknesses and reuse the advantages of the two methods discussed above, we propose a new method called **Fast Semantic Filtering**. The aim of this method is to effectively and efficiently deal with large scale ontology matching. Basically, this method adopt the efficient local optimization method of ALCOMO, which is based on incomplete reasoning and local optimal diagnosis methods in order to detect conflict pairs of mappings and remove the inconsistent mappings.

**Algorithm 5:** FAST SEMANTIC FILTERING METHOD

```
input  : $O_1$, $O_2$ : input ontologies,
         $A$ : initial alignment
output: $A_c$ : coherent alignment
DescendingSortByConfidenceValues($A$);
$A_c \leftarrow \emptyset$
while $\|A\| > 0$ do
    $m \leftarrow$ extractFirstMapping($A$);
    if isNotConflict($m, A_c, O_1, O_2$) then
        $A_c \leftarrow A_c \cup \{m\}$
    end
end
```

The main steps of this method is shown in the Algorithm 5. It first sorts all mappings in the initial alignment $A$ by confidence values in descending order. Then, for each iteration, it picks up the mapping with highest confidence value from the initial alignment $A$ into the semantic verification process. 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 $A_c$.

Obviously, in this method, confidence values of mappings and conflict patterns are the most important. To adapt this method to the large scale ontology matching task, the following modifications and extensions of the ALCOMO and ASMOV approaches should be added:

- A new and fast method based on a new structural indexing technique is used for checking a specific relationship between two concepts instead of using an existing reasoning system. It is important because any reasoning system works very slow and requires a lot of memory to classify a big ontology. Moreover, there exist many cases, in which a reasoner cannot finish its classification process even when the size of ontology is small [66].

- New heuristics are used to detect more possible conflict mappings. Note that, ALCOMO is based on the explicit existence of disjoint axioms. If the input ontologies have no or very small number of disjoint axioms, the ALCOMO may bypass inconsistent mappings.

- A new confidence propagation method is used to enhance the reliability of confidence values of candidate mappings. It is important because all of the selection methods used in ASMOV and ALCOMO are highly sensitive to the order of confidence values

## Structural Indexing

First of all, we present an example in Fig.6.9 for illustrating how to check if two concepts are disjoint.

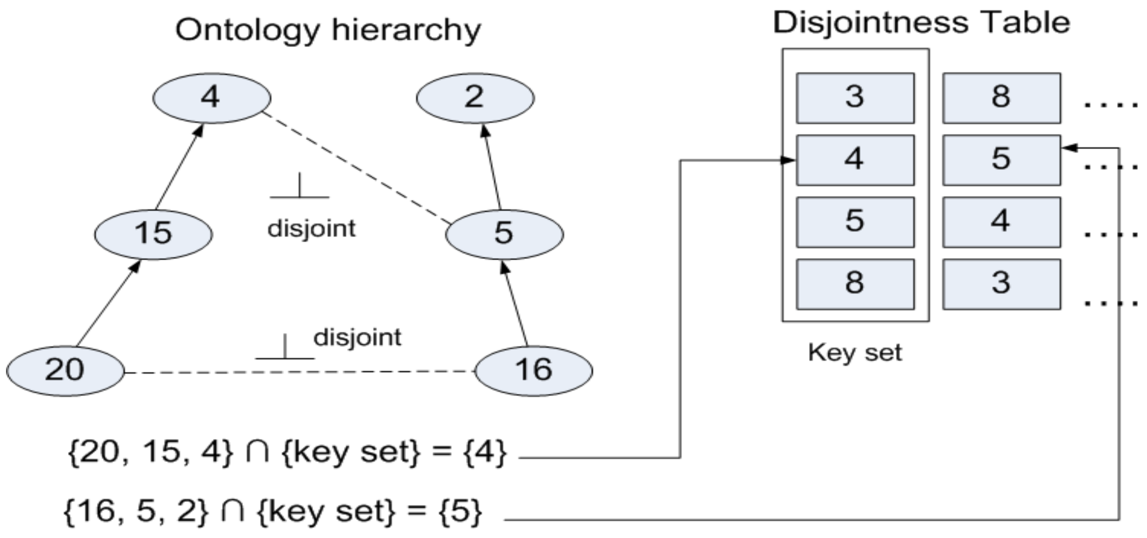


Figure 6.9: Example of checking two concepts being disjoint

In Fig. 6.9, among ancestors of concept 20, concept 4 is found in the disjoint table. Moreover, among the set of concepts, which disjoint with concept 4, concept 5 is found in ancestors of concept 16. Therefore, concepts 20 and 16 are disjoint. That means the subsumption and disjoint propagation patterns can be easily checked with help of the *ancestors* function, which returns ancestors of a given concept in a given ontology. Similarly, the other patterns like *criss cross correspondences*, *subsumption and equivalence incompleteness* can be easily checked if the *ancestors* function is available. Therefore, we believe that the efficiency of the *ancestors* function is the key for efficient detection of conflict patterns.

In this section, we introduce our method for indexing the hierarchy of ontology that allows us fast and easily verify the relationships of any two concepts. Basically, our method is implemented by using a bitmap compression method and leads to *super speed* ($O(1)$ time) of some key queries on relations among concepts such as finding their ancestors, finding the lowest common ancestor of two concepts. Based on the structural indexing, the propagation patterns used in ALCOMO and other patterns in ASMOV can be easily implemented.

Let us explain Algorithm 6, after having topologically sorted $G$ ($O(|V| + |E|)$), we implement a bitmap index compression method to store ancestors information in order to quickly perform queries in getting the relationships between entities and minimize CPU usage. The basic idea of bitmap indexes is to assign one bitmap

**Algorithm 6:** Indexing Ontology Structure

```
input  : $O$ : an ontology,
         $isa$ : subsumption relation
output: $M$ : Bitset codings of all concepts
1 $G = (V, E) \leftarrow$ Transform($O$);
2 $V_o \leftarrow$ TopologicalSort($G$, isa);
3 for $i \leftarrow 1$ to $|V_o|$ do
4     $M_i \leftarrow$ newBitSet();
5     $M_i$.set($i$);
6     forall the $v_j \mid$ isa($v_j, v_i$) do
7         $M_i \leftarrow M_i$.OR($M_j$);
8     end
9 end
```

(*i.e.,* bit array) to each entity. Each bitmap has length $|V_o|$; initially, all bitmap values are set to 0. Then, in topological order, if entity $i$ is an ancestor of $j$, the $i$-th component of the bitmap of $j$ will be set to 1. Without compression, bitmap indexes are impractically large and the processing is time consuming. Here we use compression techniques Enhanced Word-Aligned Hybrid (EWAH) based on run-length encoding (RLE) published in [57].

In terms of storage bound, these techniques help to store the bitmap index in $O(|V_o|\, c)$ where $c$ is number of bit 1 in a bitmap in average. In computational bound, by using these techniques, logical operations over any two compressed bitmaps $B_1$, $B_2$ are in $O(|B_1| + |B_2|)$ where $|B_1|, |B_2|$ are proportional to the number of bit 1 in corresponding basic bitmaps. Moreover, we observe that the maximum depth of some very big ontology is very small with respect to the number of entities. For example, according to Bioprotal[1], the maximum depth of SNOMED is 32, the maximum depth of GALEN is 26, the maximum depth of FMA is 22. This means that the number of bits 1 in basic bitmaps is relatively small in comparison with the length of bitmaps. Consequently, only $O(1)$ time is needed to perform further AND, OR operations over any two compressed bitmaps.

In particular, we directly benefit advantages of this indexing for the following queries:

- Concept $A$ is a subclass of concept $B$ if a bit at the position corresponding to index number of $B$ in the topo order is set to 1 in the bitmap of $A$. Then this query can perform in $O(1)$ time.

- To find the common ancestors of two concepts whose bitmaps are $B_1$ and $B_2$,

[1]http://bioportal.bioontology.org/

we compute bitmap $B \leftarrow B_1\, \mathbf{AND}\, B_2$ in $O(1)$. The common ancestors are concepts whose index number in the topo order is equal to the position of bit 1 in $B$. Especially, one of the most interesting thing here is that the lowest common ancestor of $B_1, B_2$ is the concept whose index is the last bit 1 in $B$. Obviously, the time complexity for this query is also $O(1)$.

## New Heuristic for Detecting Conflict

As we discussed above, ALCOMO strongly depends on the existence of disjoint axioms in the ontologies being matched. If the disjoint axioms are omitted (e.g., there is no disjoint axiom in the two ontologies of the Library track in OAEI 2012), ALCOMO cannot detect conflict mappings. In that case, similar to ALCOMO, ASMOV cannot detect any conflict based on the *disjoint subsumption contradiction* pattern. One of solution to overcome this problem is that we can use the *criss cross correspondences* pattern proposed in ASMOV. Therefore, we use this pattern as one of complement to the ALCOMO patterns.

In addition, we propose a heuristic about disjoint between two concepts without explicit declaration in a large scale ontology. The idea is that *if the similarity of two concepts in a given ontology is too small, we can consider that two concepts are disjoint*. Therefore, the problem of detecting the disjoint of two concepts can be transformed into the problem of computing similarity value between them.

Generally, computing similarity value of two concepts in an ontology is similar to computing similarity value of two synsets in the Wordnet dictionary. As we mentioned in section 3.2.2, there are many similarity measures working on Wordnet and each of them has its own strength and weakness. In order to select the most suitable measure for our problem, let see an illustration in the Fig.6.10.

Obviously, if concept $A$ and concept $B$ are disjoint, their descendants are disjoint also. That means the disjoint between two concepts does not depend of the distance of the path from one concept to the other. For example, the distance of two concept $A$ and $B$ is equal 2, but the distance of two concept $X$ and $Y$, which are descendants of $A$, $B$ respectively, maybe much longer than 2. From this observation, we assume that the disjoint of two concepts depends of the information content of their lowest common ancestor (e.g., the concept $C$ in the Fig.6.10). Therefore, in this case, we can apply the Resnik [82] measure to compute the similarity value of two concepts in ontology. We propose a definition of relative disjoint between two concepts in an ontology as follows:

**Definition 13.** ***Relative disjoint***. *Two concepts a, b in a given ontology O are relative disjoint if $ResnikSim(a, b) < \theta$, where $\theta$ is a small value in range [0, 0.1].*

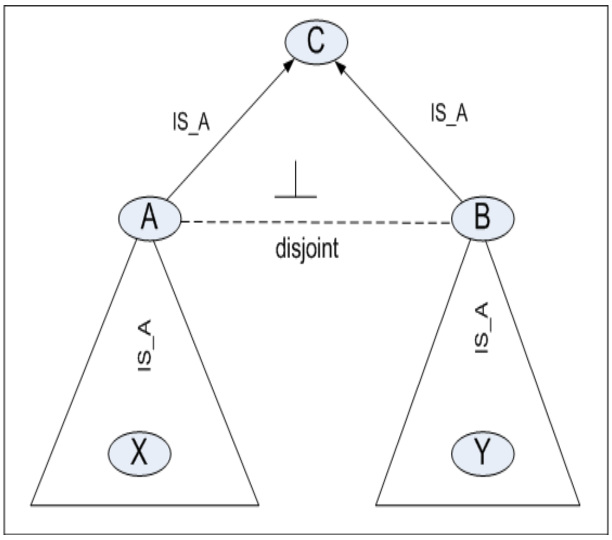


Figure 6.10: An illustration for our heuristic of disjoint concepts

Note that, the Resnik similarity computation of two senses in Wordnet is based on the notion of information content. In fact, the similarity value of two concepts is equal to the information content of a concept, which is the lowest common ancestor of the two concepts in the ontology. However, what is the information content (IC *for short*) of a concept in a given ontology? In computation linguistic, IC is a fundamental dimension measuring the amount of information such as the degree of generality or concreteness provided by the concept when appearing in the context. The basic idea is that general and abstract concepts convey less IC than the concrete and specialized ones in a given discourse. Classical information theoretic approaches [82] obtain the IC values by statistically analyzing corpora. The IC of a concept is defined as the inverse of its appearance probability in a given corpus. The IC value is then obtained by considering the negative log likelihood:

$$IC(a) = -\log p(a)$$

However, the drawback of this computation is that the probability depends of the size and background of input copora. The content of corpora should be adequate with respect to the domain knowledge and big enough in order to avoid data sparseness. Moreover, due to the scalability problem such as manual tagging and copora dependency and availability, this kind of IC computation is not applicable.

To overcome those limitation, in recent years, new methods have been proposed to compute IC of concepts from an ontology in an intrinsic manner [85]. In fact, those new methods are derived from the classical ones. Here, the intuition of intrinsic IC is that, in a well organized hierarchy (especially in an ontology), the meaning of a concept is not only encoded by itself, but also inherited from its descendants. Therefore, counting the number of descendants of a concept may refer to its appearance frequency of the domain knowledge represented by a given ontology.

In relation to our task, we realize that most of the ontologies are well organized. In order to compute the IC of concepts on anatomy ontology, we extend the state of the art intrinsic IC model proposed in [88]. This model is based on two assumptions as follows:

- Concepts with many leaves in the hierarchy are general (*i.e.*, they have low IC) as they subsume the meaning of many salient concepts.

- Concept inheriting from several subsumers makes it more specific than another one inheriting from a unique subsumer, even belonging to the same level of depth, as the former incorporates differential features from several concepts.

To sum up, based on hierarchy of is-a hierarchy, the IC value of a concept $a$ is defined as:

$$IC(a) = -\log\left(\frac{\frac{|leaves(a)|}{subsumers(a)} + 1}{max\_leaves + 1}\right)$$

From the indexing of the ontology structure, we can easily get the sets $leaves(a)$ and $subsumers(a)$ with $O(n)$ time complexity. In reality, one concept may have only several children and parents, so the real time complexity is approximately $O(1)$.

Now, we can restrict the *equivalence incompleteness* and *multiple-entity correspondences* patterns described in the ASMOV with condition of relative disjoint as follows.

**Definition 14.** ***Relative disjoint Conflict****.* *Two correspondences* $\langle a_1, a_2\rangle$ *and* $\langle b_1, b_2\rangle$ *are found by this pattern if* $O_1 \models a_1 \equiv a_2$ *but* $b_1$ *and* $b_2$ *are relative disjoint in* $O_2$ *.*

The purpose of this pattern is to specialize in large scale ontology matching, which allows multiple matching cardinality. In that case, one concept of an ontology may be matched with many concepts of another ontology. It is only acceptable if the matched concepts in the target ontology are semantically close to each other. If they are relatively disjointed, then some of them are incorrect.

### Confidence Propagation

As we discussed in the ASMOV and ALCOMO approaches, the result of semantic selection strongly depends of the confidence values of the mappings being examined.

Basically, a confidence value is used to express the degree of truth in the correctness of a mapping. When conflict mappings are found, the mapping with the lowest confidence is removed. Therefore, the certainty of confidence values should be as rigorous as possible. Ideally, the distinction between the confidence values of a correct and an incorrect mapping should be clear. That is the confidence value of an incorrect mapping is smaller than the confidence value of a correct one when they both are found in conflict mappings.

In order to improve the accuracy of our semantic selection method, we propose a method named confidence propagation, which aims to increase the confidence values of the correct mappings and to decrease the confidence values of the incorrect mappings. The heuristic of this method is that the confidence value of a mapping becomes more certain if it can find other mappings whose entities have the same semantic relationships with its entities.

**Algorithm 7:** Confidence Propagation

```
input : $O_1$, $O_2$ : input ontologies,
        $A$ : alignment
output: $A'$ : alignment with updated confidence values
for $a = \langle e_1, e_2, \equiv, c\rangle \in A$ do
    $c' = c +$ getContextProfileSim$(e_1, e_2)$;
    $a \leftarrow \langle e_1, e_2, \equiv, c'\rangle$;
end
for $a = \langle e_1, e_2, \equiv, c'\rangle \in A$ do
    $AE_1 \leftarrow$ ancestors$(O_1, e_1)$;
    $AE_2 \leftarrow$ ancestors$(O_2, e_2)$;
    forall the $ae = \langle ae_1, ae_2, \equiv, ac'\rangle \in A \mid ae_1 \in AE_1 \wedge ae_2 \in AE_2$ do
        $a \leftarrow \langle e_1, e_2, \equiv, c' + ac'\rangle$;
        $ae \leftarrow \langle ae_1, ae_2, \equiv, ac' + c'\rangle$;
    end
end
$A' \leftarrow$ normalizedConfidenceValues$(A)$
```

The idea of confidence propagation is presented in the Algorithm 7. It is similar to the idea of the similarity propagation method described in the section 4.3. However, in this method, a confidence value is only propagated from one mapping to the other in the given alignment instead of the whole pairwise connectivity graph. Here, at the line 2, function *getContextProfileSim* returns the similarity value of two concepts by comparing their context profiles as we discussed in the section 3.3.2. The benefit of using this measure is that even though the neighbor concepts around two similar concepts in the two input ontologies may be not pairwise similar, but the vocabulary used to describe them may be highly overlapped. It is especially important for the large scale ontology matching, where the number of correct mappings is too small with respect to the size of ontologies. For example, FMA and NCI ontologies contain 78,989 and 66,724 classes respectively, but they share only about 2,900 concepts. That means the distribution of the correct mappings seems very sparse.

Next, after updating the similarity value by comparing the context profile, the propagation process propagates the new similarity values (i.e., new confidence values) of all mappings in the given alignment among their subsumption path in the ontology hierarchies (line 5 to line 12). At lines 6 and 7, function *ancestors* returns all ancestors of a concept in a given ontology. At line 13, function $normalizedConfidenceValues$ normalizes the confidence values of mappings into range of $[0, 1]$.

## 6.3 Experiments and Evaluations

In this section, we evaluate the performance of our semantic filtering method. Because our contribution focus on the large scale ontology matching, in the following experiment, we are going to compare the performance of our method (i.e., Fast Semantic Filtering) and the methods provided in ALCOMO tool on the Anatomy and Biomedical ontology matching (i.e., Small FMA-NCI, Large FMA-NCI, Whole FMA-NCI) data sets used in OAEI 2011.5. Note that, the size of the whole FMA and NCI ontologies are 78,989 and 66,724 classes, respectively. The size of the mouse and human ontologies in the Anatomy test are 2744 and 3304 classes, respectively.

Our prototype performing this experiment consists of three main components.

- The first component is based on Label Filter, which is an efficient filter to select the high possible candidate mappings. The description of this filter can be read in section 7.3.3.
- The second component is based on Information retrieval based similarity measure described in section 3.3.1
- The last component is based on either a method in ALCOMO tool or our fast semantic filtering method. In terms of ALCOMO, we use the incomplete and efficient method $ALCOMO_{IE}$.

Running both ALCOMO and our methods in the same prototype assures the fairness in comparison. Table 6.1 and Table 6.2 show the results obtained by applying ALCOMO and our method, respectively. The $ALCOMO_{IE}$ method only won our method in case of $LargeFMA - NCI$ test, whereas, our method won it in the rest

| Test set | Precision | Recall | Fmeasure | Run times |
|---|---|---|---|---|
| Anatomy | 0.9227 | 0.87401 | 0.8977 | 154 (s) |
| Small FMA - NCI | 0.95896 | 0.88682 | 0.92148 | 1082 (s) |
| Large FMA - NCI | 0.73065 | 0.86301 | 0.79133 | 2281 (s) |
| Whole FMA - NCI | - | - | - | - (s) |

Table 6.1: Performance of $ALCOMO_{IE}$ on large scale ontology matching

| Test set | Precision | Recall | Fmeasure | Run times |
|---|---|---|---|---|
| Anatomy | 0.944 | 0.868 | 0.904 | 201 (s) |
| Small FMA - NCI | 0.980 | 0.848 | 0.9093 | 482 (s) |
| Large FMA - NCI | 0.923 | 0.821 | 0.869 | 1908 (s) |
| Whole FMA - NCI | 0.906 | 0.821 | 0.861 | 3864 (s) |

Table 6.2: Performance of the fast semantic filtering on large scale ontology matching

tests. Especially, in case of $WholeFMA - NCI$, ALCOMO did not pass due to an exception of out of memory. In fact, we have tested ALCOMO with other large scale tests with SNOMED ontology (122464 classes) but it failed. In terms of our method, we passed all tests including SNOMED ontology. The other results returned by our method can be read in section 8.3.1.

## 6.4 Conclusion

In this chapter, we have presented different mapping selection methods that are widely used in ontology matching systems. We have categorized them into two groups, i.e., non-semantic selection methods and semantic selection methods.

Our contribution in this chapter focuses on semantic selection method. For this purpose, we have analyzed semantic verification methods used in ASMOV and ALCOMO tools. In the analyses, we have shown the strengths and weaknesses of each method.

In terms of ASMOV, we have argued that some conflict patterns of ASMOV are not rigorous. Moreover, the verification method used in ASMOV may be error prone due to the uncertainty of similarity values computed in each running iteration of ASMOV.

In terms of ALCOMO, we have argued that it maybe not work well if the input ontologies do not have any disjoint axiom. In that case, it cannot detect any conflict pair of mappings. Moreover, ALCOMO strongly depends of the use of reasoning system (i.e., Pellet). In many cases where Pellet cannot classify one of the input ontologies, this tools will fail.

To overcome the weaknesses of ASMOV and ALCOMO, we have proposed a **Fast Semantic Filtering** method to refine the discovered mappings for large scale ontology matching. In our method, we have proposed a heuristic of *Relative disjoint*, which can find to disjoint concepts in a big ontology. It can be useful in the case when there is no disjoint axiom number declared in an ontology or the number of this axiom is too small. Moreover, we have proposed an efficient method to index the structure of a ontology. By using this indexing method, we can efficiently verify the relation of any pair of entities.

Finally, the experimental result shows that our method can work even in very large scale ontology matching, whereas, the incomplete method in ALCOMO cannot. Furthermore, our method outperforms ALCOMO both in matching quality and performance time.

# Chapter 7

# Towards Large Scale Ontology Matching

Matching large scale ontologies is one of the most difficult problems in the 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. It is because large ontologies usually lead to very high conceptual heterogeneity, which makes the ontology matching system difficult to find all mappings. Consequently, the effectiveness of the matching system will be decreased. Furthermore, large ontologies produce a huge search space, in which a matching system seeks all correct mappings. A discovery mappings in the 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 decreased also.

In this chapter, we mainly focus on the approaches dealing with large scale ontology matching. For this purpose, we overview first several existing methods used in the state of the art ontology matching systems. In particular, a Partition-based method will be discussed in section 7.1. Then, an Anchor-based Segmentation method will be discussed in section 7.2. In the discussion, we will show the strengths and weaknesses of these methods.

Our contribution in this chapter is new methods for reducing the search space. In particular, in section 7.3, we propose two heuristics for selecting the candidate mappings that are likely to be matched. The first heuristic states that if two entities of two ontologies are likely similar if the context information of one entity can be found in the context information of the other. This heuristic leads us to design two filtering methods, namely **Description Filter** and **Context Filter**, which are based on *search engine* techniques. These methods will be discussed in section 7.3.1 and section 7.3.2, respectively. On the other hand, the second heuristic states that if two entities are likely similar, their labels are likely similar too. Based on this idea, we propose a filtering method called **Label Filter**. This method will be presented in section 7.3.3. Finally, in section 7.4, we present our experiment for evaluating the performance of our filtering methods.

## 7.1 Partition-based Method

In this section, we discuss the main features of a partition-based method dealing with large scale ontology matching. Fig. 7.1 depicts the main steps of the partition-based ontology matching method. The algorithm used in this method is similar to the divide and conquer algorithm. Here, a big problem (i.e., matching on large ontologies) is broken down into smaller problems (i.e., matching on sub ontologies).

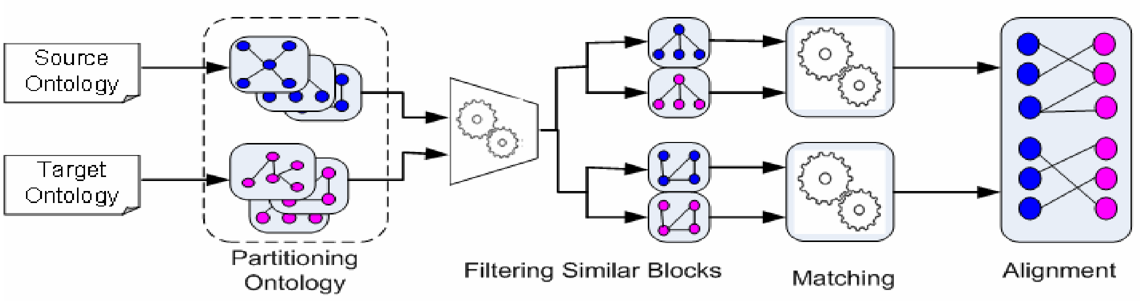


Figure 7.1: Partition-based ontology matching

In this method, two main sequent 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 works [3, 40], 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 represented different and independent sub-domains of knowledge.

- Next, in order to reduce the run-time complexity, 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 computed by a similarity measure (e.g., string-based measure). For example, this method is implemented in [40]. Let $B_1$ and $B_2$ be two sets of blocks from two ontologies $O_1$ and $O_2$ and Q be a set of anchors found between $O_1$ and $O_2$. Function $TAnchors(b_1, b_2, Q)$ returns the total number of the anchors in Q between two blocks $b_1$ and $b2$ ($b_1 \in B_1$, $b_2 \in B_2$). The similarity between $b_1$ and $b_2$ is defined as follows:

$$sim(b_1, b_2) = \frac{2 \cdot TAnchors(b_1, b_2, Q)}{\sum_{b_i \in B_1} TAnchors(b_i, b_2, Q) + \sum_{b_j \in B_2} TAnchors(b_1, b_j, Q)}$$

  By setting a cutoff $\theta \in [0..1]$, any two blocks whose similarity is greater than $\theta$ is selected to make up a block mapping.

- 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 [3]. The computation of similarity values between two block documents is similar to the computation of similarity between two virtual document in section 3.3.2.

The advantages of this method is that is can be used as a upper layer of existing matching methods which can produce a high matching quality but 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 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.

## 7.2 Anchor-based Segmentation Method

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 similarity values. This method can be found in QOM [?], Anchor-PROMPT [72] systems, and more recently in the Anchor-Flood [38] system which obtained the best runtime in the anatomy track in OAEI 2008.

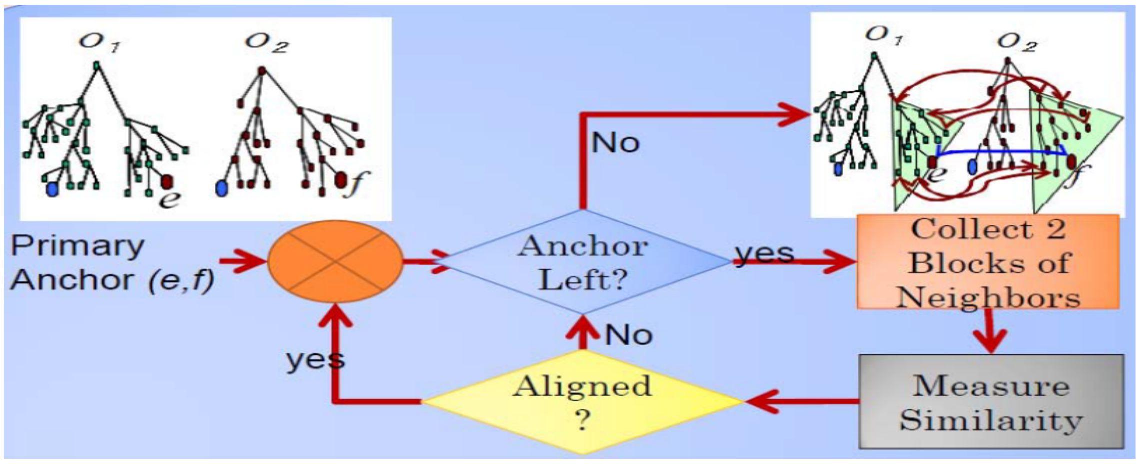


Figure 7.2: Dynamic segmentation with anchor

Fig.7.2 illustrates the dynamic segmentation method in the Anchor-Flood system. Basically, the main steps of this method work as follows:

- Firstly, at least one of initial anchor 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 subconcepts of certain depth. The intuition is that the neighbors of similar concepts might also be similar.

- Then, the aligning 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 runs 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 large and the positions 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 [46], 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 aligning 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.

## 7.3 Similar Annotation Oriented Heuristics

In this section, we are going to present our heuristics to deal with large scale ontology matching, in particular with biomedical ontologies. The proposed heuristics are inspired from two observations described in the work [32] as follows.

- Firstly, biomedical ontologies are usually large in size and have relatively little structure, with only a few relationships. Moreover, real-world large ontologies are highly conceptual heterogeneity in terms of different properties such as coverage, granularity and perspective. Therefore, algorithms that rely heavily on analyzing the structure do not have an advantage.

- Second, biomedical ontologies often have rich terminological information, with many synonyms specified for each concept. Therefore, just using labels and synonyms as a source for mappings provides good results. Moreover, there is probably less variability in the language used to name biomedical concepts compared to other domains. In fact, in order to distinguish thousands of concepts, the developers of biomedical ontologies should precisely select and assign labels to each concept. All these factors make lexical techniques effective.

These observations may be right not only for matching with large ontologies in the biomedical field, but also for matching large scale ontologies in general. Besides, as we mentioned in chapter 4, discovery matching from conceptual heterogeneity is more complex than discovery matching from terminological heterogeneity. Moreover, its matching quality also strongly depends on the matching quality found by terminological matcher. Therefore, in our approach, we try first to discover as many as possible numbers of mappings by exploiting terminological information of entities. Then, the structural and semantic information of entities are exploited to discover new mappings and to verify the correctness of each mapping.

In order to discover mappings by using terminological information, i.e., annotation of entities, we proposed two heuristics as follows:

- Two entities are similar if their **textual context** are highly similar. The textual context of an entity consists of a vocabulary used to describe its semantic meaning in the modeling domain.

- Two entities are similar if their name or labels or synonyms differ by maximum two tokens. The intuition is that if two labels of two entities differ by more than three tokens, any string-based similarity measure will produce a low similarity score value. Then, these entities are highly unmatched.

Based on these two heuristics, we have designed three filters for selecting candidates named Description Filter, Context Filter and Label Filter.

### 7.3.1 Filter by Description

The aim of this filter is to fast locate pairs of entities from the input ontologies, which have highly similar descriptions. Here, the **description** of an entity is a simple textual context, which is created from name, labels, synonyms and comments in the annotation of the entity. In fact, the description of an entity is usually a long text. Using an exhaustive algorithm to compare every pair of entities from the input ontologies will be very time consuming. Therefore, in this filter, we propose a fast method based on search engine technique for selecting candidate mappings. The two main steps of this method work as follows:

- For the first input ontology (source ontology), Lucene search engine indexes and stores the description of each entity. Lucene search engine automatically tokenizes the description into tokens; performs token-stemming and stores them in its indexing directory.

- For each entity in the second ontology (target ontology), a multi token-query is created from the collection of tokens in the description. The query is then executed within the Lucene indexing directory to return a sorted list of results. Each searching result consists of an entity of the source ontology and a ranking score, which shows the similarity between the description of this entity and the query. From the list of results, we select a top-K highest results to form a top-K candidates.

This method is useful when entities in both input ontologies have some comments in their annotation.

### 7.3.2 Filter by Context

This filter is an extension of the Description Filter discussed above. For each entity in ontology, instead of a simple description, its context is composed of 3 independent descriptions such as:

- An **internal description** consisting of name, labels, synonyms and comments in the annotation of the entity.

- An **ancestor description** consisting of a collection of internal descriptions of ancestors of the entity.

- A **descendant description** consisting of a collection of internal descriptions of descendant of the entity.

The intuition behind this method is that if two entities of the two input ontologies are similar, we can find that the vocabulary used to describe their ancestors, descendants and themselves are similar. This assumption leads to three main steps as follows:

- At the indexing step, for each entity in the source ontology, Lucene search engine indexes and stores 3 independent descriptions in 3 independent indexing directories.

- At the searching step, for each entity in the target ontology, three independent queries constructed from its three descriptions are executed in the corresponding indexing directory. For example, a query created from ancestor description is executed in the indexing directory containing ancestor descriptions of all entities in the source ontology. Thus, for each entity in the target ontology, the Lucene search engine returns three lists of results corresponding to an ancestor query, an internal query and a descendant query. Note that, each result consists of an entity of the source ontology and a corresponding ranking score.

- At the selection step, it selects the **top-K** entities from the three lists of results, which has the highest sum of ranking scores.

In fact, this method is similar to the filtering method described in [11], which obtains a good result in the Anatomy track ($Fmeasure = 0.88$). However, in our approach, thanks to Lucene, which is one of the fastest state of the art search engine, this filter provides good performance.

### 7.3.3 Filter by Label

The aim of this filter is to fast locate pairs of entities from the input ontologies, which have highly similar name, labels or synonyms. The selected pairs of entities are considered as candidate mappings, which will be judged by a string similarity measure.

As we mentioned in our proposed heuristics, if two labels differ by more than three non-stop words (tokens), their similarity computed by any string measure is low. Therefore, two entities in a candidate mapping should have at least a pair of labels that differ by less than two non-stop words.

Based on this idea, the main steps of this method are as follows:

- For each input ontology, build an inverted map, where the key of each entry is a normalized label and the value is the entity of the ontology. The procedure of producing normalized labels for an entity will be demonstrated in the example below.

- Extract the two key sets of the two inverted maps and get their intersection. Obviously, each item in the intersection set is a shared label between some entities of the input ontologies. Those entities can be easily obtained by a look up of the values corresponding to a shared label in the inverted maps.

The efficiency of this method lies in the way of manipulating appropriate data structure. Here, thanks to the hash function, the intersection operation between two set of labels runs very fast. Moreover, thanks to inverted map data structure, it is fast to get the entities by a given label key.

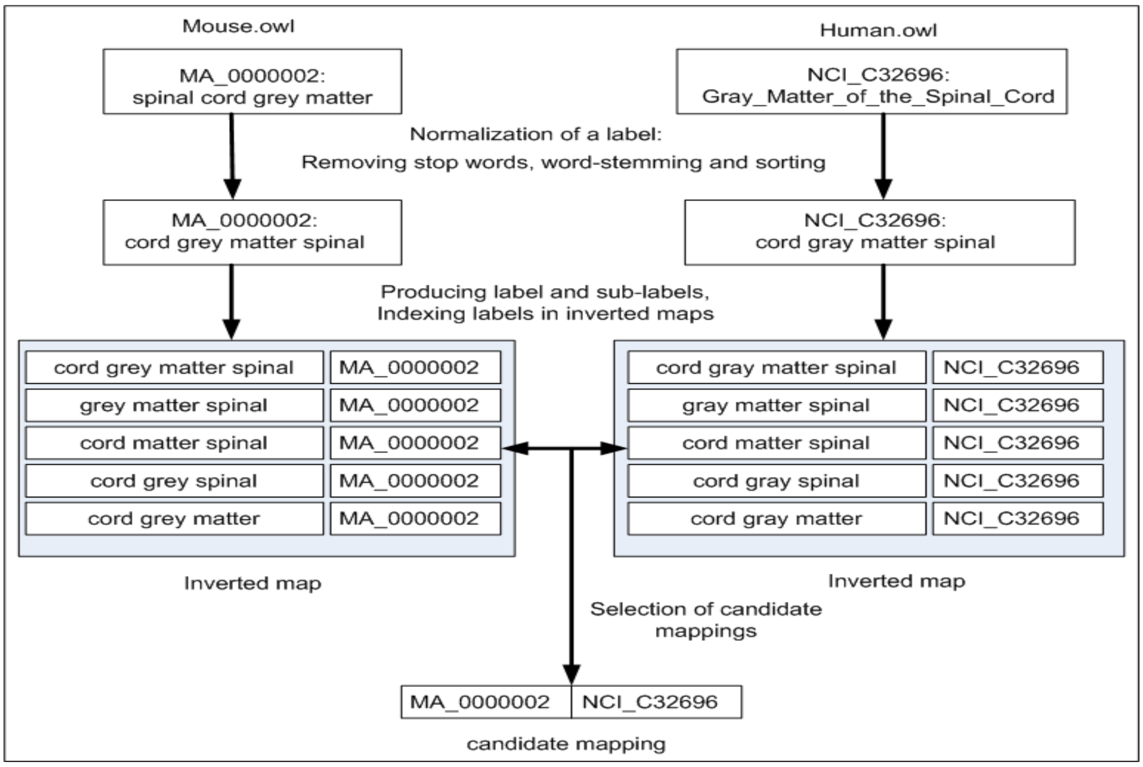

Mouse.owl
Human.owl
MA_0000002: spinal cord grey matter
NCI_C32696: Gray_Matter_of_the_Spinal_Cord
Normalization of a label: Removing stop words, word-stemming and sorting
MA_0000002: cord grey matter spinal
NCI_C32696: cord gray matter spinal
Producing label and sub-labels, Indexing labels in inverted maps
cord grey matter spinal | MA_0000002
grey matter spinal | MA_0000002
cord matter spinal | MA_0000002
cord grey spinal | MA_0000002
cord grey matter | MA_0000002
cord gray matter spinal | NCI_C32696
gray matter spinal | NCI_C32696
cord matter spinal | NCI_C32696
cord gray spinal | NCI_C32696
cord gray matter | NCI_C32696
Inverted map
Inverted map
Selection of candidate mappings
MA_0000002 | NCI_C32696
candidate mapping

Figure 7.3: An example with the label filtering method

Fig. 7.3 illustrates a simple example extracted from the anatomy track in OAEI2012. Obviously, the labels of two entities are not identical, but highly similar. Thanks to the Label Filter, the two entities are detected as a candidate mapping very fast. Then, our Information Retrieval based similarity measure described in section 3.3.1 is used to compute the similarity value of the two entities by comparing their original labels, i.e., "*spinal cord grey matter*" vs. "*Gray_Matter_of_the_Spinal_Cord*".

## 7.4 Experiments and Evaluations

In this section, we evaluate the performance of our proposed filters. Two criteria considered in this evaluation are *Recall* and *Runtime*. Here, *Recall* indicates the

| **Top-K** | **Precision** | **Recall** | **F-Measure** | **TP** | **FP** | **FN** | **Runtime** |
|---|---|---|---|---|---|---|---|
| $K = 1$ | 0.432 | 0.778 | 0.556 | 1179 | 1546 | 337 | 23 (s) |
| $K = 2$ | 0.398 | 0.781 | 0.527 | 1184 | 1790 | 332 | 23 (s) |
| $K = 3$ | 0.231 | 0.847 | 0.363 | 1285 | 4268 | 231 | 23 (s) |
| $K = 4$ | 0.216 | 0.855 | 0.345 | 1296 | 4694 | 220 | 23 (s) |
| $K = 5$ | 0.158 | 0.889 | 0.269 | 1343 | 7122 | 173 | 23 (s) |
| $K = 6$ | 0.150 | 0.887 | 0.257 | 1345 | 7604 | 171 | 24 (s) |
| $K = 7$ | 0.121 | 0.908 | 0.214 | 1377 | 9972 | 139 | 24 (s) |
| $K = 8$ | 0.117 | 0.910 | 0.207 | 1380 | 10437 | 136 | 24 (s) |
| $K = 9$ | 0.099 | 0.923 | 0.178 | 1400 | 12775 | 116 | 24 (s) |
| $K = 10$ | 0.095 | 0.926 | 0.173 | 1404 | 13293 | 112 | 24 (s) |

Table 7.1: Performance of the context filtering method on the Anatomy test

| **Top-K** | **Precision** | **Recall** | **F-Measure** | **TP** | **FP** | **FN** | **Runtime** |
|---|---|---|---|---|---|---|---|
| $K = 1$ | 0.610 | 0.802 | 0.693 | 2262 | 1444 | 557 | 111 (s) |
| $K = 2$ | 0.333 | 0.876 | 0.483 | 2471 | 4940 | 348 | 111 (s) |
| $K = 3$ | 0.229 | 0.902 | 0.365 | 2544 | 8570 | 275 | 112 (s) |
| $K = 4$ | 0.174 | 0.915 | 0.293 | 2581 | 12235 | 238 | 112 (s) |
| $K = 5$ | 0.141 | 0.925 | 0.245 | 2609 | 15908 | 210 | 117 (s) |
| $K = 6$ | 0.118 | 0.929 | 0.209 | 2620 | 19598 | 199 | 120 (s) |
| $K = 7$ | 0.102 | 0.936 | 0.183 | 2638 | 23281 | 181 | 120 (s) |
| $K = 8$ | 0.089 | 0.939 | 0.163 | 2648 | 26972 | 171 | 125 (s) |
| $K = 9$ | 0.080 | 0.944 | 0.147 | 2661 | 30659 | 158 | 125 (s) |
| $K = 10$ | 0.072 | 0.947 | 0.134 | 2670 | 34349 | 149 | 125 (s) |

Table 7.2: Performance of the context filtering method on the FMA-NCI-small test

number of correct mappings can be discovered by other matchers, and *Runtime* indicates about the efficiency of our filtering methods.

Generally, the higher *Recall* is, the more mappings can be obtained. Besides, *Precision* in this experiment is less important than *Recall*. It is because the aim of our methods is to find a subset of entities from one ontology that possibly match to one entity of the other ontology. Many incorrect mappings maybe also involved in the result of our methods, consequently, the *Precision* may be low. However, improving *Precision* is the job of similarity matchers.

In this experiment, we use dataset of the **Anatomy** and **Biomedical ontology matching** tracks in OAEI 2012. The Anatomy test consists of a mouse ontology with 2744 classes and a human ontology with 3304 classes. A test taken from the Biomedical ontology matching track is FMA-NCI-small, which consists of a small fragment of FMA ontology with 3696 classes and small fragment of NCI ontology with 6488 classes.

Table 7.1 and Table 7.2 show the performance of the Context Filter on the

| Test | Precision | Recall | F-Measure | TP | FP | FN | Runtime |
|---|---|---|---|---|---|---|---|
| **Anatomy** | 0.922 | 0.872 | 0.896 | 1322 | 112 | 194 | 14 (s) |
| **FMA-NCI-small** | 0.912 | 0.899 | 0.905 | 2534 | 243 | 285 | 20 (s) |

Table 7.3: Performance of the label filtering method on the Anatomy and FMA-NCI-small tests

Anatomy and the FMA-NCI-small tests. Note that this method makes use of the Lucene Search Engine to search the most similar entities to a given entity by comparing their contexts. Parameter $K$ implies that for each entity in the query, the top $K$ entities of the searching result for each query will be selected. In this experiment, the value $K$ is varied from 1 to 10.

The general trend of both tables shows that when the value $K$ increases, the *Recall* increases also. For example, when $K$ is set to 10, *Recall* for the Anatomy test is 0.926 and *Recall* for the FMA-NCI-small test is 0.947. The most important thing is that the number of candidate mappings significantly decreases. For example, the total pairs of entities coming from two ontologies in the Anatomy test are: $2744 \cdot 3304 = 9066176$, whereas, after filtering, the total candidate mappings is only $1404 + 13293 = 14697$. Moreover, the runtime of the filtering process is very small (maximum 24 seconds for the Anatomy test).

Table 7.3 shows the performance of the Label Filter on the Anatomy and FMA-NCI-small tests. Surprisingly, it run fast (about 14 seconds and 20 seconds for each test) and obtained very high quality. In particular, for the Anatomy test, the *Fmeasure* is 0.896, and for the FMA-NCI-small test, the *Fmeasure* is 0.905. These results prove that our heuristic proposed in the Section 7.3.3 is efficient.

## 7.5 Conclusion

In this chapter, we have presented our methods to deal with large scale ontology matching. In particular, we have only focused on the improvement of efficiency. In our approach, two heuristic for candidate filtering have been proposed, which implies that two entities are likely similar if their contexts and their labels are highly similar, respectively.

The first heuristic have been used in the two filtering methods, namely Description Filter and Context Filter in section 7.3. These methods make use of the Lucene search engine to indexing and searching contexts of entities in the input ontologies. The experiment in section 7.4 shows that the performance of these methods depends on the parameter $K$, which implies how many entities will be selected from the query result. The higher value of $K$ is, the higher *Recall* will be obtained.

The second heuristic have been implemented in the Label Filter in section 7.3. The experiment in section 7.4 shows that this filter is very effective and efficient. Therefore, to deal with large scale biomedical ontology matching, we can use this method to reduce the search space before running the main matching process.

# Chapter 8

# Ontology Matching with YAM++

In this chapter, we present our ontology matching system called YAM++, which have implemented all the contributions discussed in the previous chapters. In order to evaluate the performance of our approach, we perform experiments with the standard data sets published in the OAEI campaigns and compare the results of YAM++ with other participants in the campaigns.

In section 8.1, we first introduce the prototype and evaluation results of the first version (YAM++ v.1.0), which participated to the OAEI 2011 campaign. Next, in section 8.2, we present the modification that had been made from YAM++ v.1.0 to the YAM++ v.1.5. In addition, we present the comparison result of YAM++ v.1.5 with other participants in OAEI 2011.5 campaign. Section 8.3 presents the current version YAM++ v.2.0. This version is able to deal with large scale ontology matching. In this section, we present the performance of YAM++ v.2.0 in both terms of matching quality and runtime efficiency when it deals with large scale data sets.

## 8.1 YAM++ v.1.0 in OAEI 2011 Campaign

This is the first version of YAM++ that participated to the campaign OAEI 2011. The motivation of this version is to deal with the *selection and configuration tuning challenge* in the ontology matching field. That is, two main issues should be considered such as (i) selection of appropriate individual matchers (or similarity measures), and (ii) combination with self-tuning configuration for the selected matchers. The idea for solving these issues in the whole matching system is that a divide and conquer method will be used to deal with these issues in each component of the system.

Fig. 8.1 shows the main components of YAM++ v.1.0. The **Parsing & Pro-**

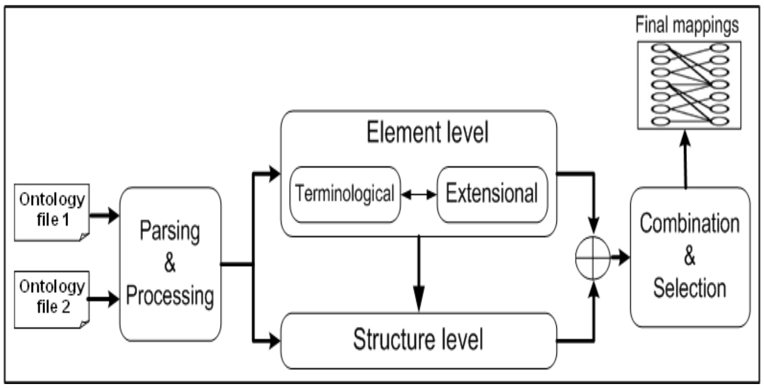


Figure 8.1: Main components of YAM++ v.1.0

**cessing** component is used to read and load the input ontologies into the internal data structure in the main memory. This component is built up on the OWLAPI library and Pellet reasoner.

The matching process is performed within the two main components i.e., **element level matcher** and **structure level matcher**. The element level matcher is mainly based on a **terminological matcher**, which discovers mappings of entities by comparing their annotation informations (i.e., labels, comments). In order to deal with high terminological heterogeneity, several terminological similarity measures are combined in this matcher. In our approach, we propose a machine learning based methods to select and combine those similarity measures. This method is discussed in detail in Section 5.2 in Chapter 5. Besides, an **extentional matcher** described in Section 3.3.3 in Chapter 3 complements the matching results to the **terminological matcher**. The combination between them is simply performed by making an union of their results. Finally, we have an element level matching result.

On the other hand, the structure level matcher is based on the similarity propagation method, which is described in detail in Section 4.3 in Chapter 4. The output of the element level matcher is passed as input to this matcher in order to perform a similarity propagation process. In turn, this matcher produces a structure level matching result.

In order to combine the matching results of both element and structure level matchers, we apply the **dynamic weighted sum** method which is described in Section 5.3 in Chapter 5. Here, the Greedy selection method, which is described in the Section 6.1.2 in Chapter 6, is used to produce the final alignment.

| **Configuration** | **Precision** | **Recall** | **F-Measure** |
|---|---|---|---|
| **ML** | 0.99 | 0.72 | 0.84 |
| **ML + IB** | 0.99 | 0.74 | 0.85 |
| **ML + IB + SP** | 0.98 | 0.84 | 0.90 |

Table 8.1: Evaluation on the Benchmark dataset based on bibliography ontology

### 8.1.1 Experiments on the OAEI 2011 Datasets

In order to demonstrate the performance of our system, we will show the effectiveness and the impact of the components above by comparing the results with following configurations. The basic configuration consists of the Terminological matcher only, which is based on a machine learning (**ML**) approach. Next, we extend it with the Extensional component, which is an Instance based matcher (**IB**), in order to have a full Element-level matcher. Finally, we add the Similarity Propagation (**SP**) component to the third configuration. All experiments are executed with JRE 6.0 on Intel 3.0 Pentium, 3Gb of RAM, 1Gb for JVM in Window XP SP3.

#### Benchmark

The first benchmark data set, which is based on bibliography ontology, includes 111 tests. Each test consists of source (reference) ontology and a test ontology, which is created by altering some information from the reference. The reference ontology contains 33 named classes, 24 object properties, 40 data properties, 56 named individuals and 20 anonymous individuals. The evaluation of our system on this track with the three configurations is shown in the Table 8.1.

The observation from this track is as follows. By using only machine learning (**ML**) approach, YAM++ achieved good result with very high precision (**0.99**) and F-Measure (**0.84**). After adding the Instance based (**IB**) matcher, both Recall and F-Measure increased with **2%** and **1%** respectively. These improvements were happened because many ontologies have common extensional data (instances). Finally, thanks to the process of propagation of similarity, both Recall and F-Measure increased with **10%** and **5%** respectively.

The second benchmark data set based on conference ontology includes 103 tests. Similar to the biblio benchmark, in this track, a source (original) ontology is compared to target ontologies which were obtained by altering some features from the original. The reference ontology contains 74 named classes, 33 object properties without extensional data (instances). The evaluation of our system on this track with the three configurations is shown in the Table 8.2.

Similar to the Benchmark 2010 track, using Similarity Propagation method in-

| Configuration | Precision | Recall | F-Measure |
|---|---|---|---|
| **ML** | 0.98 | 0.51 | 0.67 |
| **ML + IB** | 0.98 | 0.51 | 0.67 |
| **ML + IB + SP** | 0.97 | 0.60 | 0.74 |

Table 8.2: Evaluation on the Benchmark dataset based on conference organization ontology

| Configuration | Precision | Recall | F-Measure |
|---|---|---|---|
| **ML** | 0.75 | 0.50 | 0.60 |
| **ML + IB** | 0.75 | 0.50 | 0.60 |
| **ML + IB + SP** | 0.78 | 0.56 | 0.65 |

Table 8.3: Evaluation on the Conference dataset

creases both Recall and F-Measure values with **9%** and **7%**, respectively. The Instance based matcher did not improve the performance because in this track, ontologies don't support extensional data. That is why the matching results of running the first and the second configurations are the same.

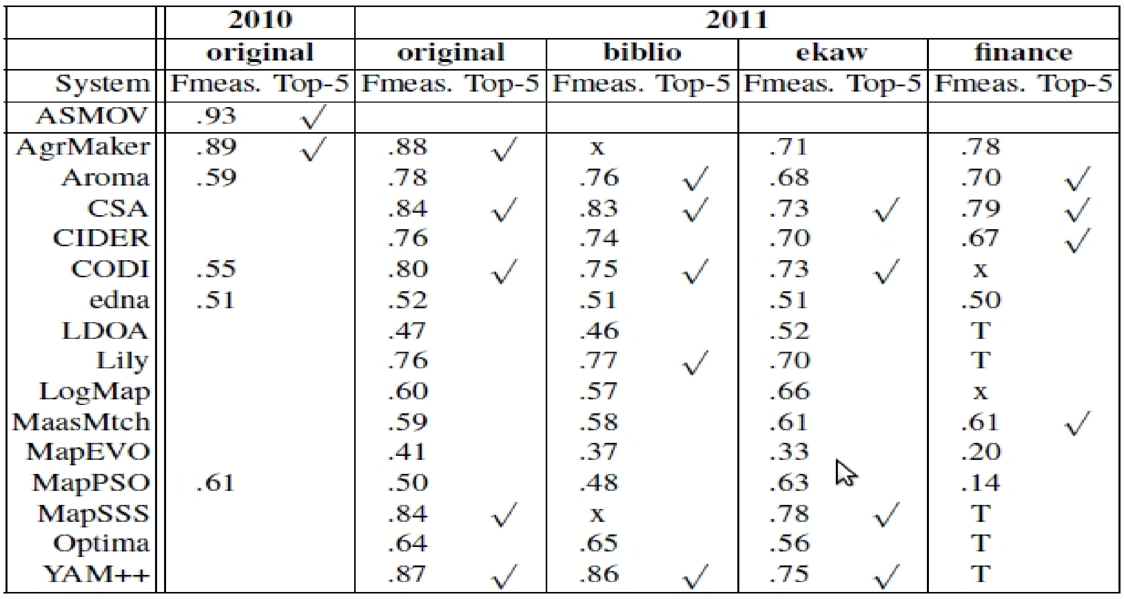
**Conferences**

Conference track now contains 16 ontologies from the same domain (conference organization) and each ontology must be matched against every other ontology. Due to the high heterogeneity of those ontologies, finding mappings between them is more difficult than that in benchmark tracks. Besides, this track is an open+blind test because there are no reference alignments for most of those tests. In the Table 8.3, we can only report our results with respect to the available reference alignments. The observation on this track is similar to the second benchmark track. Here, thanks to our Similarity Propagation method, all of Precision, Recall and F-Measure values are improved.

### 8.1.2 YAM++ vs. Other Participants in OAEI 2011

In order to compare the performance of YAM++ v.1.0 with other systems, we submitted YAM++ to the SEALS portal. To participate to the campaign, all participants have to select the best configuration for their performances. It is a very good condition of the campaign that avoids a manual tuning. The organizers of the SEALS portal run all participated system on the common data sets and fairly made comparison of performance between those systems.

In the OAEI 2011 campaign, three tracks i.e., **Benchmark**, **Conference** and



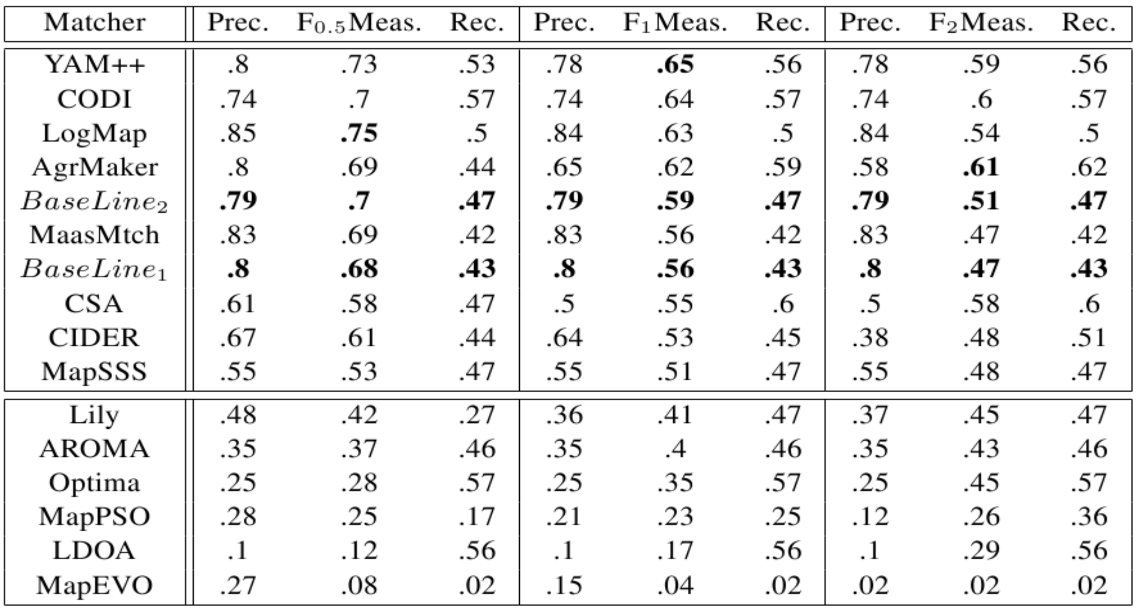
| | 2010 | | 2011 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | original | | original | | biblio | | ekaw | | finance | |
| System | Fmeas. | Top-5 | Fmeas. | Top-5 | Fmeas. | Top-5 | Fmeas. | Top-5 | Fmeas. | Top-5 |
| ASMOV | .93 | ✓ | | | | | | | | |
| AgrMaker | .89 | ✓ | .88 | ✓ | x | | .71 | | .78 | |
| Aroma | .59 | | .78 | | .76 | ✓ | .68 | | .70 | ✓ |
| CSA | | | .84 | ✓ | .83 | ✓ | .73 | ✓ | .79 | ✓ |
| CIDER | | | .76 | | .74 | | .70 | | .67 | ✓ |
| CODI | .55 | | .80 | ✓ | .75 | ✓ | .73 | ✓ | x | |
| edna | .51 | | .52 | | .51 | | .51 | | .50 | |
| LDOA | | | .47 | | .46 | | .52 | | T | |
| Lily | | | .76 | | .77 | ✓ | .70 | | T | |
| LogMap | | | .60 | | .57 | | .66 | | x | |
| MaasMtch | | | .59 | | .58 | | .61 | | .61 | ✓ |
| MapEVO | | | .41 | | .37 | | .33 | | .20 | |
| MapPSO | .61 | | .50 | | .48 | | .63 | | .14 | |
| MapSSS | | | .84 | ✓ | x | | .78 | ✓ | T | |
| Optima | | | .64 | | .65 | | .56 | | T | |
| YAM++ | | | .87 | ✓ | .86 | ✓ | .75 | ✓ | T | |

Figure 8.2: Comparison of YAM++ with other participants in the Benchmark track in the OAEI 2011 (taken from [1])

**Anatomy** were used for evaluation. Fig.8.2 and Fig.8.3 show the comparison of YAM++ v.1.0 with other participants on the benchmark and conference tracks.

In terms of the Benchmark track, YAM++ achieved a position in **top-2** best matching systems with respect to the **original**, **biblio** and **ekaw** data sets. In particular, YAM++ obtained `0.87` of *Fmeasure* and stood behind AgreementMaker (`0.88` of *Fmeasure*) for the original data set. For the **biblio** data set, YAM++ were the best tool with **0.86** of *Fmeasure*. For the **ekaw** data set, YAM++ obtained **0.75** of *Fmeasure* and only lost MapSSS, which obtained **0.78** of *Fmeasure*. However, YAM++ did not pass the **Finance** data set in the Benchmark track, which were designed for large scale matching. It was the same problem of YAM++ with the **Anatomy** track.

In terms of the Conference track, YAM++ achieved a position in **top-2** best matching systems with respect to different computation of *Fmeasure*. In particular, when the weight of *Precision* is higher than that of *Recall* (in case of $F_{0.5}measure$), YAM++ lost LogMap system. When the weight of *Recall* is higher than that of *Precision* (in case of $F_2measure$, YAM++ lost CODI and AgreementMaker. But, in terms of harmonic mean of precision and recall ($F_1measure$), YAM++ obtained the best result.



| Matcher | Prec. | $F_{0.5}$Meas. | Rec. | Prec. | $F_1$Meas. | Rec. | Prec. | $F_2$Meas. | Rec. |
|---|---|---|---|---|---|---|---|---|---|
| YAM++ | .8 | .73 | .53 | .78 | **.65** | .56 | .78 | .59 | .56 |
| CODI | .74 | .7 | .57 | .74 | .64 | .57 | .74 | .6 | .57 |
| LogMap | .85 | **.75** | .5 | .84 | .63 | .5 | .84 | .54 | .5 |
| AgrMaker | .8 | .69 | .44 | .65 | .62 | .59 | .58 | **.61** | .62 |
| $BaseLine_2$ | **.79** | **.7** | **.47** | **.79** | **.59** | **.47** | **.79** | **.51** | **.47** |
| MaasMtch | .83 | .69 | .42 | .83 | .56 | .42 | .83 | .47 | .42 |
| $BaseLine_1$ | **.8** | **.68** | **.43** | **.8** | **.56** | **.43** | **.8** | **.47** | **.43** |
| CSA | .61 | .58 | .47 | .5 | .55 | .6 | .5 | .58 | .6 |
| CIDER | .67 | .61 | .44 | .64 | .53 | .45 | .38 | .48 | .51 |
| MapSSS | .55 | .53 | .47 | .55 | .51 | .47 | .55 | .48 | .47 |
| Lily | .48 | .42 | .27 | .36 | .41 | .47 | .37 | .45 | .47 |
| AROMA | .35 | .37 | .46 | .35 | .4 | .46 | .35 | .43 | .46 |
| Optima | .25 | .28 | .57 | .25 | .35 | .57 | .25 | .45 | .57 |
| MapPSO | .28 | .25 | .17 | .21 | .23 | .25 | .12 | .26 | .36 |
| LDOA | .1 | .12 | .56 | .1 | .17 | .56 | .1 | .29 | .56 |
| MapEVO | .27 | .08 | .02 | .15 | .04 | .02 | .02 | .02 | .02 |

Figure 8.3: Comparison of YAM++ with other participants in the Conference track in the OAEI 2011 (taken from [1])

### 8.1.3 Conclusion of YAM++ v.1.0 Version

In this section, we presented the prototype of the first version of YAM++, which focus on the selection and configuration tuning challenge in the ontology matching. The experimental results showed that individual components of YAM++ work effectively. Furthermore, the comparison results showed that YAM++ is high competitive with other state of the art ontology matching systems.

However, there were two drawbacks of this version. Firstly, it based on a machine learning model to combine similarity measures for discovering mappings. The performance of the learning model strongly depends on the given training data. In case that the training data are not available or are not suitable for a given matching scenario, the performance of YAM++ will be low.

Besides, the second drawback relates to large scale matching. In the first time participated to the campaign, YAM++ v.1.0 did not pass any test in Finance and Anatomy data sets. The main reason is because a very big propagation graph in the similarity propgation method, that lead to the problem of overload of the main memory. Moreover, storing temporary data for the learning and classifying processes of the machine learning model requires a lot of memory also. Another reason is because of using Pellet reasoner to classify the input ontologies in the parsing and processing step. A very large ontology is very time and memory resource consuming to perform a complete reasoning task. In some case, if the input ontologies are incoherent, Pellet cannot finish its job.

## 8.2 YAM++ v.1.5 in OAEI 2011.5 Campaign

Generally, the architecture of this version and the first version are the same. Several changes had been made to this version as follows:

- Firstly, due to the high complexity and due to the problem of lacking training data for the machine learning based method, in the version YAM++ v.1.5, we removed the machine learning part and used the **information retrieval similarity measures** described in section 3.3.1 in Chapter 3.
- Secondly, in order to semantically refine the discovered mappings, we made use of a debugging tool for incoherent mappings, namely **ALCOMO** [66].
- Lastly, due to the OAEI 2011.5 campaign provided multilingual ontology matching track, we incorporated a multilingual translator, i.e., **Microsoft Bing**[1], in this version in order to translate labels of entities from other languages to English.

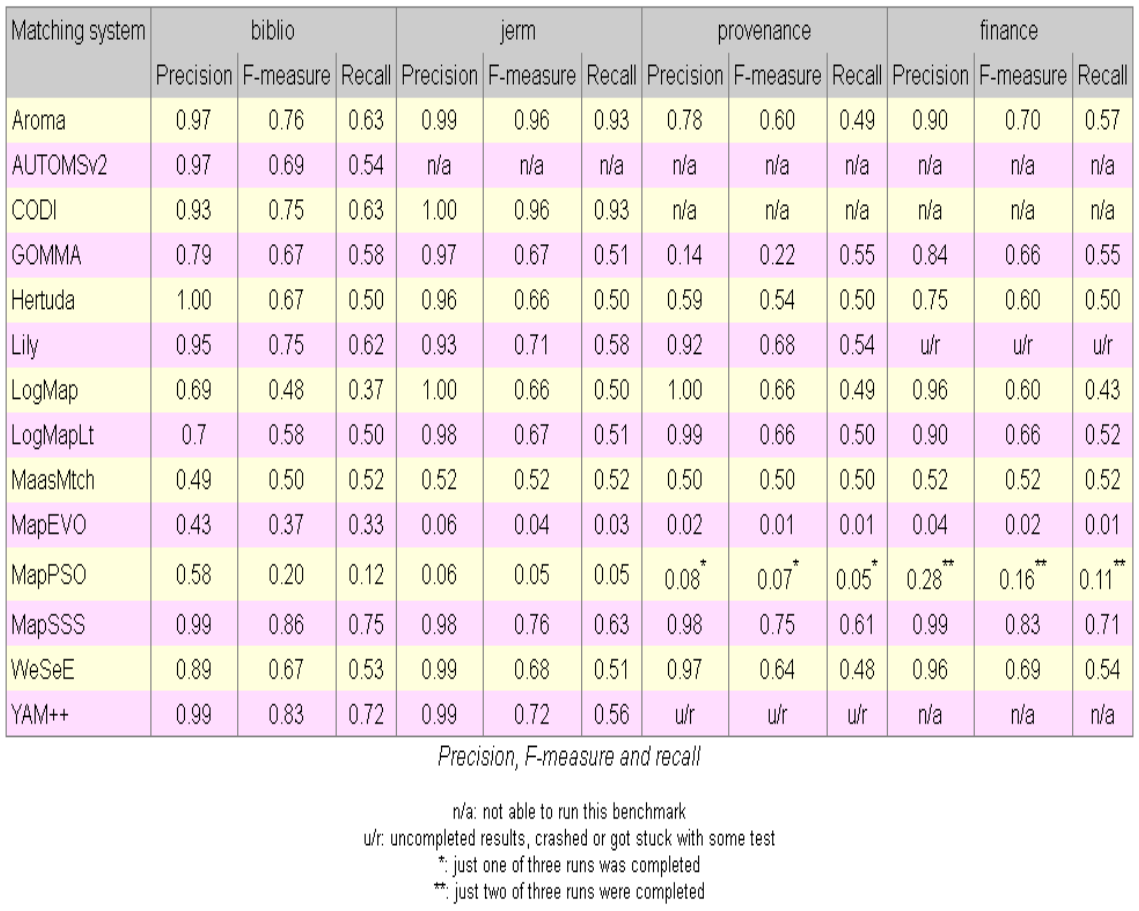
However, because of the short time between the two campaigns, the problem of large scale matching was still retained in this version. Therefore, in the OAEI 2011.5 campaign, YAM++ v.1.5 was able to produce results for the Benchmark, Conference and Multifarm tracks only.

### 8.2.1 YAM++ vs. Other Participants in OAEI 2011.5

The comparisons of YAM++ v.1.5 and other participants are shown in the Fig. 8.4, Fig.8.5 and Fig.8.6.

In the OAEI 2011.5 campaign, new data sets were added to the Benchmark track. According to the increasing sizes (total number classes and properties) of the seed ontologies from **biblio** (97) to **jerm** (205), **provenance** (431) and **finance** (633), the Benchmark track became a scalability track. In this track, YAM++ v.1.5 produced good result on the **biblio** data set with 0.83 of *Fmeasure*. YAM++ got the second position on this data set, where the best tool were MapSSS with 0.86 of *Fmeasure*. For the **jerm** data set, YAM++ stood in the top 4 best tools with *Fmeasure* is equal to 0.72. The best tools were CODI and AROMA obtained a surprisingly high *Fmeasure*, which is equal to 0.96. In terms of the **provenance** and the **finance** data sets, YAM++ did not complete or did not pass the whole tests due to problems related to the lacking of memory.

[1]http://www.bing.com/translator

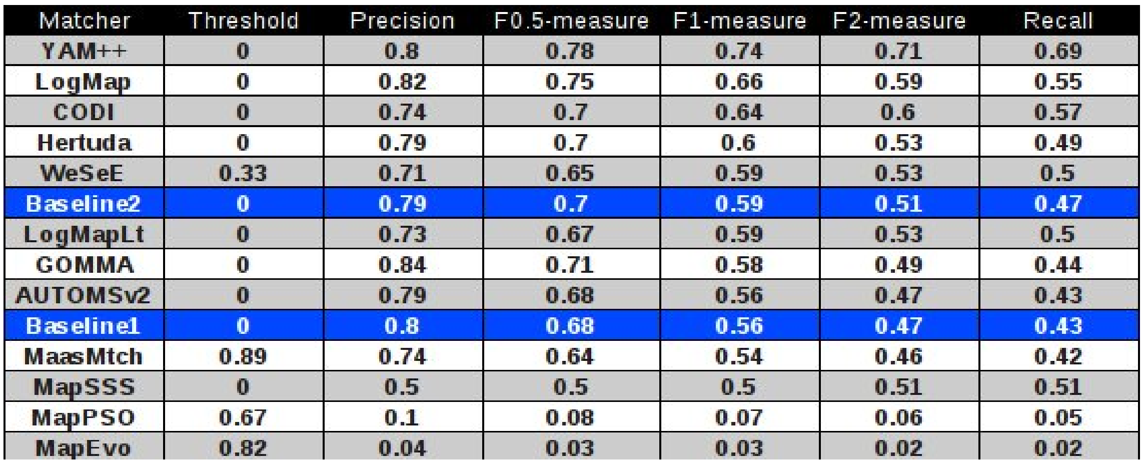


| Matching system | biblio | | | jerm | | | provenance | | | finance | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | F-measure | Recall | Precision | F-measure | Recall | Precision | F-measure | Recall | Precision | F-measure | Recall |
| Aroma | 0.97 | 0.76 | 0.63 | 0.99 | 0.96 | 0.93 | 0.78 | 0.60 | 0.49 | 0.90 | 0.70 | 0.57 |
| AUTOMSv2 | 0.97 | 0.69 | 0.54 | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a |
| CODI | 0.93 | 0.75 | 0.63 | 1.00 | 0.96 | 0.93 | n/a | n/a | n/a | n/a | n/a | n/a |
| GOMMA | 0.79 | 0.67 | 0.58 | 0.97 | 0.67 | 0.51 | 0.14 | 0.22 | 0.55 | 0.84 | 0.66 | 0.55 |
| Hertuda | 1.00 | 0.67 | 0.50 | 0.96 | 0.66 | 0.50 | 0.59 | 0.54 | 0.50 | 0.75 | 0.60 | 0.50 |
| Lily | 0.95 | 0.75 | 0.62 | 0.93 | 0.71 | 0.58 | 0.92 | 0.68 | 0.54 | u/r | u/r | u/r |
| LogMap | 0.69 | 0.48 | 0.37 | 1.00 | 0.66 | 0.50 | 1.00 | 0.66 | 0.49 | 0.96 | 0.60 | 0.43 |
| LogMapLt | 0.7 | 0.58 | 0.50 | 0.98 | 0.67 | 0.51 | 0.99 | 0.66 | 0.50 | 0.90 | 0.66 | 0.52 |
| MaasMtch | 0.49 | 0.50 | 0.52 | 0.52 | 0.52 | 0.52 | 0.50 | 0.50 | 0.50 | 0.52 | 0.52 | 0.52 |
| MapEVO | 0.43 | 0.37 | 0.33 | 0.06 | 0.04 | 0.03 | 0.02 | 0.01 | 0.01 | 0.04 | 0.02 | 0.01 |
| MapPSO | 0.58 | 0.20 | 0.12 | 0.06 | 0.05 | 0.05 | 0.08* | 0.07* | 0.05* | 0.28** | 0.16** | 0.11** |
| MapSSS | 0.99 | 0.86 | 0.75 | 0.98 | 0.76 | 0.63 | 0.98 | 0.75 | 0.61 | 0.99 | 0.83 | 0.71 |
| WeSeE | 0.89 | 0.67 | 0.53 | 0.99 | 0.68 | 0.51 | 0.97 | 0.64 | 0.48 | 0.96 | 0.69 | 0.54 |
| YAM++ | 0.99 | 0.83 | 0.72 | 0.99 | 0.72 | 0.56 | u/r | u/r | u/r | n/a | n/a | n/a |

*Precision, F-measure and recall*

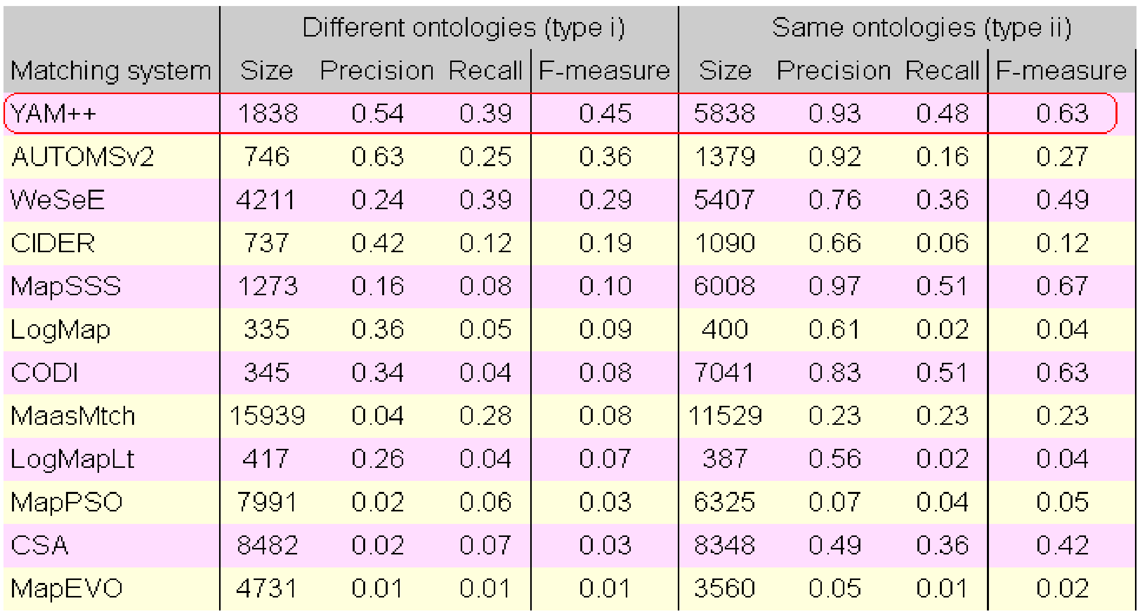
n/a: not able to run this benchmark
u/r: uncompleted results, crashed or got stuck with some test
*: just one of three runs was completed
**: just two of three runs were completed

Figure 8.4: Comparison of YAM++ with other participants in the Benchmark track in the OAEI 2011.5 (taken from [2])



| Matcher | Threshold | Precision | F0.5-measure | F1-measure | F2-measure | Recall |
|---|---|---|---|---|---|---|
| YAM++ | 0 | 0.8 | 0.78 | 0.74 | 0.71 | 0.69 |
| LogMap | 0 | 0.82 | 0.75 | 0.66 | 0.59 | 0.55 |
| CODI | 0 | 0.74 | 0.7 | 0.64 | 0.6 | 0.57 |
| Hertuda | 0 | 0.79 | 0.7 | 0.6 | 0.53 | 0.49 |
| WeSeE | 0.33 | 0.71 | 0.65 | 0.59 | 0.53 | 0.5 |
| Baseline2 | 0 | 0.79 | 0.7 | 0.59 | 0.51 | 0.47 |
| LogMapLt | 0 | 0.73 | 0.67 | 0.59 | 0.53 | 0.5 |
| GOMMA | 0 | 0.84 | 0.71 | 0.58 | 0.49 | 0.44 |
| AUTOMSv2 | 0 | 0.79 | 0.68 | 0.56 | 0.47 | 0.43 |
| Baseline1 | 0 | 0.8 | 0.68 | 0.56 | 0.47 | 0.43 |
| MaasMtch | 0.89 | 0.74 | 0.64 | 0.54 | 0.46 | 0.42 |
| MapSSS | 0 | 0.5 | 0.5 | 0.5 | 0.51 | 0.51 |
| MapPSO | 0.67 | 0.1 | 0.08 | 0.07 | 0.06 | 0.05 |
| MapEvo | 0.82 | 0.04 | 0.03 | 0.03 | 0.02 | 0.02 |

Figure 8.5: Comparison of YAM++ with other participants in the Conference track in the OAEI 2011.5 (taken from [2])



| | Different ontologies (type i) | | | | Same ontologies (type ii) | | | |
|---|---|---|---|---|---|---|---|---|
| Matching system | Size | Precision | Recall | F-measure | Size | Precision | Recall | F-measure |
| YAM++ | 1838 | 0.54 | 0.39 | 0.45 | 5838 | 0.93 | 0.48 | 0.63 |
| AUTOMSv2 | 746 | 0.63 | 0.25 | 0.36 | 1379 | 0.92 | 0.16 | 0.27 |
| WeSeE | 4211 | 0.24 | 0.39 | 0.29 | 5407 | 0.76 | 0.36 | 0.49 |
| CIDER | 737 | 0.42 | 0.12 | 0.19 | 1090 | 0.66 | 0.06 | 0.12 |
| MapSSS | 1273 | 0.16 | 0.08 | 0.10 | 6008 | 0.97 | 0.51 | 0.67 |
| LogMap | 335 | 0.36 | 0.05 | 0.09 | 400 | 0.61 | 0.02 | 0.04 |
| CODI | 345 | 0.34 | 0.04 | 0.08 | 7041 | 0.83 | 0.51 | 0.63 |
| MaasMtch | 15939 | 0.04 | 0.28 | 0.08 | 11529 | 0.23 | 0.23 | 0.23 |
| LogMapLt | 417 | 0.26 | 0.04 | 0.07 | 387 | 0.56 | 0.02 | 0.04 |
| MapPSO | 7991 | 0.02 | 0.06 | 0.03 | 6325 | 0.07 | 0.04 | 0.05 |
| CSA | 8482 | 0.02 | 0.07 | 0.03 | 8348 | 0.49 | 0.36 | 0.42 |
| MapEVO | 4731 | 0.01 | 0.01 | 0.01 | 3560 | 0.05 | 0.01 | 0.02 |

Figure 8.6: Comparison of YAM++ with other participants in the Multifarm track in the OAEI 2011.5 (taken from [2])

For the Conference track, thanks to our new proposed similarity measure, which is based on Information retrieval techniques, and thanks to the semantic verification component, which is based on the ALCOMO tools, the overall results of YAM++ significantly increased. In particular, this version increased 0.13 $(0.69 - 0.56)$ in $Recall$ and 0.02 $(0.80 - 0.78)$ in $Precision$. Therefore, the $Fmeasure$ of this version were increased 0.09 $(0.74 - 0.65)$ in comparison with the first version. Furthermore, in this track, YAM++ v.1.5 obtained value of 0.74 for $Fmeasure$ and outperformed other participants. The second best tool were Logmap, which obtained 0.66 for $Fmeasure$.

The goal of the MultiFarm track is to evaluate the ability of matcher systems to deal with multilingual ontologies. It is based on the OntoFarm dataset, where annotations of entities are represented in different languages such as: English (en), Chinese (cn), Czech (cz), Dutch (nl), French (fr), German (de), Portuguese (pt), Russian (ru) and Spanish (es). For those tests of this track, thanks to a multilingual translator, which were used to translate annotations of entities in those ontologies into English, YAM++ were able to discover alignments between them. According to the comparison results, YAM++ were the winner on this track. In particular, there were two types of evaluation. In the first type, all matching tools dealt with different ontologies in different languages. In this evaluation, YAM++ achieved the best matching quality (Fmeasure = 0.45). In the second type, all tools discovered mappings of the same ontologies but translated in different languages. In this evaluation, YAM++ obtained the second position among all participants.

## 8.3 YAM++ v.2.0 in OAEI 2012 Campaign

The major focus of this version (v.2.0) of YAM++ is to deal with large scale ontology matching. Especially, scalability and large scale ontology matching is the main topic of the OAEI 2012 campaign. Indeed, new data sets with very large ontologies such as Library ontologies, Biomedical ontologies have been introduced in this campaign.

In order to overcome the weakness of the last versions of YAM++, the new version have been updated and supplemented with new components. Fig.8.7 shows the main components of the YAM++ v.2.0 version.

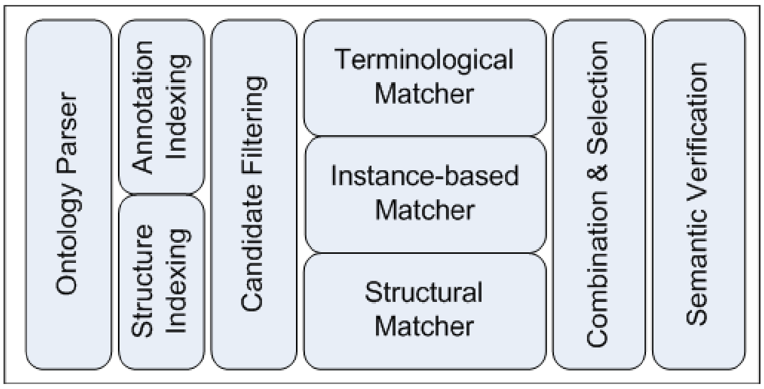


Figure 8.7: Main components of YAM++ OAEI 2012 version

In the YAM++ v.2.0, a generic workflow for a given ontology matching scenario is as follows.

1. Input ontologies are loaded and parsed by the **Ontology Parser** component;

2. Information of entities in ontologies are indexed by the **Annotation Indexing** and the **Structure Indexing** components.

3. **Candidates Filtering** component filters out all possible pairs of entities from the input ontologies, whose descriptions are highly similar;

4. Among those candidate mappings, the **Terminological Matcher** component produces a set of mappings by comparing the annotations of entities;

5. The **Instance-based Matcher** component discovers new mappings through shared instances between ontologies;

6. Like in previous versions of YAM++, matching results of the **Terminological Matcher** and the **Instance-based Matcher** are aggregated into an element level matching result. The **Structural Matcher** component then enhances the element level matching result by exploiting structural information of entities;

7. The mapping results obtained from the three matchers above are then combined and selected by the **Combination & Selection** component to have a unique set of mappings;

8. Finally, the **Semantic Verification** component refines those mappings in order to eliminate the inconsistent ones.

In this version, the **Annotation Indexing**, the **Structure Indexing** and the **Candidates Filtering** components are new; the **Semantic Verification** component have been updated with new features. The specifications of those components are described as follows.

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

**Structure Indexing** In this component, the main structure information such as IS-A and PART-OF hierarchies of ontologies are stored. This indexing method is described in detail in the Section 6.2.4 in Chapter 6. A benefit of this method is to easily access to the structure information of ontology and minimize memory for storing it. After this step, the loaded ontologies can be released to save main memory.

**Candidates Filtering** The aim of this component is to reduce the computational space for a given scenario, especially for the large scale ontology matching task. For this purpose, two filters have been designed in the YAM++ v.2.0 version. These filters are described in detail in Section 7.3 in Chapter 7.

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

Table 8.4: YAM++ results on pre-test Scalability track

| Test set | H-mean Precision | H-mean Recall | H-mean Fmeasure |
|---|---|---|---|
| Biblio | 0.98 | 0.72 | 0.83 |
| Benchmark 2 | 0.96 | 0.82 | 0.89 |
| Benchmark 3 | 0.97 | 0.76 | 0.85 |
| Benchmark 4 | 0.96 | 0.72 | 0.83 |
| Finance | 0.97 | 0.84 | 0.90 |

Table 8.5: YAM++ results on Benchmark track in OAEI 2012

**Semantic Verification** Because the ALCOMO tool has some limits in very large scale ontology matching (e.g., Biomedical track), this component is used to replace ALCOMO tool. The detail of its description can be found in the Section 6.2.4 in Chapter 6.

Thanks to these changes and modifications, YAM++ v.2.0 now can work with large scale ontology matching. At this time, we do not have the comparison results of YAM++ with other participants in the OAEI 2012. We will present the evaluation results of YAM++ with data sets published in the OAEI 2012 campaign, in particular, the scalability and large scale data sets will be focused. Here, all experiments are executed with JRE 6.0 on Intel 3.0 Pentium, 3Gb of RAM, 1Gb for JVM in Window XP SP3.

## 8.3.1 Experiments on the OAEI 2012 Datasets

### Scalability track

Firstly, we evaluate the performance of YAM++ v.2.0 with the scalability tests in the OAEI 2011.5 campaign. Due to we do not have the complete results for all test sets, Table 8.4 shows the result of YAM++ running on the **pre-test**, which is not the full data set.

In OAEI 2012, **Benchmark** includes 2 open tests (i.e., biblio, finance) and 3 blind tests (i.e., Benchmark 2, 3, 4). Here, the size of the seed ontologies in those tests is 274, 354 and 472, respectively. Table 8.5 shows the results of YAM++ running on the **Benchmark** data sets:

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

Table 8.6: YAM++ results on Anatomy track

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

Table 8.7: YAM++ results on Large biomedical ontologies track

#### 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. Table 8.6 shows the evaluation result and runtime of YAM++ on this track.

#### Large Biomedical Ontologies 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 9 sub tasks with different size of input ontologies, i.e., small fragment, large fragment and the whole ontologies. Table 8.7 shows the evaluation results and runtime performance of YAM++ on those sub tasks.

### 8.3.2 Conclusion of YAM++ v.2.0 Version

Obviously, YAM++ passed all scalability and large scale ontology matching tests and obtained high matching results. Moreover, YAM++ can run with normal laptop or PC with not high configuration. That is the benefit of using structure indexing, candidate filtering and fast semantic verification components that we proposed in previous chapters.

## 8.4 Conclusion

In this chapter, we have presented the three prototypes of YAM++ system, which have participated to the OAEI 2011, OAEI 2011.5 and OAEI 2012 campaigns. We have discussed the ability and the weaknesses of each prototype. The experimental results show that the performance of YAM++ have been step by step improved. Moreover, according to the comparison results with other ontology matching systems in the OAEI campaigns, YAM++ achieved a high ranking position.

# Chapter 9

# Conclusion and Future Work

The final chapter aims to summarize the main contributions of this thesis and to outline a number of directions for future work. We start with Section 9.1 to highlight our contributions to the list of research questions initiated in the introduction of this thesis. Next, in Section 9.2, we present the remaining issues that we are concerned in future work. We focus on extensions and improvements of our approach and suggest additional experiments required to answer these issues.

## 9.1 Main Contributions

The main goal of our research is enhancing ontology matching by using techniques coming from different fields such as Machine Learning, Information Retrieval and Graph Matching. This objective was stated in the introduction with a list of five research questions. Each of these questions has been carefully studied in each chapter of this thesis in order to be effectively and completely solved.

The first issue related to the problem of dealing with terminological heterogeneity in ontology matching. Terminological heterogeneity causes many problems, in which the same entities to be labeled with different names in different ontologies. Because of the high terminological heterogeneity in the real ontologies, most of the existing terminological similarity measures are not sufficient. In order to overcome this challenge, we have designed new similarity measures based on different techniques coming from Information retrieval field. The experimental results proved that our similarity measures, i.e., Context profile measure, Instance-based similarity measure, and especially, the Information Retrieval based measure effectively deal with terminological heterogeneity with the real world ontologies. These measures are our ***first contribution***, which have been described in detail in Section 3.3. On the other hand, we have also argued that a combination of the existing similarity measures could improve the matching quality. Thus, we have proposed a combination method which is based on Machine Learning models (Section 5.2). This combination method is automatic and flexible. That is, it does not limit the number of the similarity measures to be combined and its configuration is self-tuned during the training process. The Machine Learning based combination method is our ***second contribution*** in this thesis.

The second issue related to the problem of dealing with conceptual heterogeneity in ontology matching. Conceptual heterogeneity makes the same entities have different semantic descriptions including their internal properties or their external relationships with other entities. In order to overcome this challenge, we have proposed a Similarity Propagation method, which is built on our proposed high level graph data structure for ontology. The data structure and the Similarity Propagation method are described in section 4.3. The experimental results have proved that our method is stable and less error prone than the other existing structural methods. This method and the high level graph data structure is our ***third*** contribution in this thesis.

The third issue related to the problem of combining matching results obtained from element level matcher, which may consist of several terminological matchers, and from structure level matcher. The aim of this combination is fully automatic and effective. To solve this problem, we have designed a method called Dynamic Weighted Sum in section 5.3, which automatically assigns a weigh value to each matcher. Moreover, it also automatically determines a threshold value to select the final mappings in combination. The experimental results have proved that our proposed method outperforms two other automatic weighted sum methods used in the PRIOR+ [64] and AgreementMaker [15] systems. The Dynamic Weighted Sum method is our ***fourth contribution*** in this thesis.

The fourth issue related to the problem of matching large scale ontologies. Large scale ontology matching is one of the toughest challenges in the ontology matching field. The large size of ontologies decreases both the effectiveness and the efficiency of any ontology matching system. Furthermore, it usually requires a very powerful hardware to perform the matching task. In section 7.3, we have proposed two heuristics and three new filtering methods, namely Description Filter, Context Filter and Label Filter to reduce the computational space of the matching task. The two first filters are based on the use of Search Engine technique, in particular, Lucene Search Engine. The last one is based on a *Hash* function to filter identical labels or sub-labels efficiently. These filtering methods are our ***fifth contribution***.

The last issue related to the problem of removing semantic inconsistent mappings in the large scale ontology matching. In this case, the problem of detecting inconsistent mappings becomes harder because almost all reasoning systems fail or cannot completely classify large ontologies. To overcome this challenge, we have proposed a Fast Semantic Filtering method and a new heuristic, namely Relative disjoint heuristic. Our heuristic and our filtering method have been described in section 6.2.4. The experimental results have proved that our filtering method can work with very large scale ontologies. This is our ***sixth contribution***.

Despite the fact that we have contributed a solution to each research issue, there are still many open issues that should be considered. Indeed, some of our methods are not complete, and consequently they need more investigation. In the next section, we will discuss these issues.

## 9.2 Remaining Issues and Future Work

In the following, we present some works that remains to be done. They show the possibility to improve or extend our system.

**Improving Time Performance** Efficiency is one of the main focuses that we are concerned. Despite the fact that the last version of YAM++ can deal with large and very large scale ontology matching, the runtime performance of YAM++ is low. In terms of large scale matching, the indexing of structural information of an ontology is very time consuming. It is mainly because of running topological sort overall the concepts of ontology. On the other hand, the process of semantic verification is slow because the confidence propagation is performed overall the discovered mappings. In terms of small scale matching (like Conference, Benchmark data sets), the most consuming time lies in similarity propagation method. This method is based on fix point computation, thus it runs iteratively to update all candidate mappings in the computational space.

We are now working on the optimization of the code of YAM++ in order to save memory resource and improving its efficiency. On the other hand, the current YAM++ runs with only one thread. We may extend it into multi-threading system to take advantages of multi-core processor computing system.

**Recall Improvement** For small scale ontology matching, the candidate filtering method is not used. But for the large scale matching, the candidate filtering reduces the computation space before running the main matching process. In the current version of YAM++, we use a Context Filter and Label Filter. From the experiments, we realize that the *Recall* obtained by these filters is always smaller than 1.0. That is, these filters miss a number of correct mappings.

In order to overcome this problem, we are now working on a supplementary filtering method, which is based on similar structure of entities. The intuition is similar to the idea implemented in an external structural similarity measure. A good structural indexing will help finding entities having the similar structural patterns efficiently.

Moreover, we may consider the possibility of taking into account background knowledge (like super ontology) to discover more correct mappings.

**Inconsistent Removing** In YAM++, we have designed a Fast Semantic Filtering method to eliminate inconsistent mappings. In fact, this method is a modification of the incomplete and efficient method provided in ALCOMO. In order to improve the coherent of mappings, we need to perform more experiments and maybe improve the algorithm.

**User Interaction** A graphical user interface (GUI) is necessary in order to help the user verify the discovered mappings. We are now developing a GUI that allows the user to visualize the graph of semantic context of entities for each discovered mapping. In addition, we plan to implement an incomplete reasoner to detect and show conflict mappings in the GUI. It will be a debugging tool to refine the final matching result.

On the other hand, we are concerned with user interaction. That is, we let the user the possibility to provide feedbacks on some mappings during the matching process. YAM++ will exploit the feedback in order to produce a higher matching quality.

**Instance Matching** The current version of YAM++ discovers mappings at schema level only. That is, mappings between class-class and property-property will be returned. In ontology matching, instance matching is an important task. It is because instances of an ontology are the real world data. The discovered mappings between instance-instance could be very useful in semantic data integration. Therefore, in the future, we plan to deal with instance matching.

# Bibliography

[1] OAEI 2011. http://oaei.ontologymatching.org/2011/.

[2] OAEI 2011.5. http://oaei.ontologymatching.org/2011.5/.

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

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

[5] 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.

[6] 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.

[7] Jürgen Bock, Carsten Dänschel, and Matthias Stumpp. Mappso and mapevo results for oaei 2011. In *OM*, 2011.

[8] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Comput. Linguist.*, pages 13–47, 2006.

[9] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Rec.*, 2006.

[10] Watson Wei Khong Chua and Jung-Jae Kim. Eff2match results for oaei 2010. In *OM*, 2010.

[11] Watson Wei Khong Chua and Jung-jae Kim. Boat: Automatic alignment of biomedical ontologies using term informativeness and candidate selection. *J. of Biomedical Informatics*, pages 337–349, 2012.

[12] Philipp Cimiano. *Ontology learning and population from text - algorithms, evaluation and applications.* Springer, 2006.

[13] 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.

[14] Roger L. Costello and David B. Jacobs. Owl web ontology language, 2003.

[15] Isabel F. Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. Efficient selection of mappings and automatic quality-driven combination of matching methods. In *OM*, 2009.

[16] Isabel F. Cruz and Huiyong Xiao. Using a layered approach for interoperability on the semantic web. In *WISE*, pages 221–231, 2003.

[17] Jérôme David. Aroma results for oaei 2009. In *OM*, 2009.

[18] 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.

[19] Rose Dieng and Stefan Hug. Comparison of personal ontologies represented through conceptual graphs. In *ECAI*, pages 341–345, 1998.

[20] Hong Hai Do and Erhard Rahm. Coma - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[21] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Y. Halevy. Ontology matching: A machine learning approach. In *Handbook on Ontologies*, pages 385–404. 2004.

[22] John Domingue, Dieter Fensel, and James A. Hendler, editors. *Handbook of Semantic Web Technologies.* Springer, Berlin, 2011.

[23] Fabien Duchateau, Remi Coletta, Zohra Bellahsene, and Renée J. Miller. Yam: a schema matcher factory. In *CIKM*, pages 2079–2080, 2009.

[24] Marc Ehrig. *Ontology Alignment: Bridging the Semantic Gap.* Springer, 2007.

[25] Marc Ehrig and Steffen Staab. Qom – quick ontology mapping. In *ISWC*, pages 683–697. Springer, 2004.

[26] Isabel F. Cruz et al. Using agreementmaker to align ontologies for oaei 2010. In *OM*, 2010.

[27] 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.

[28] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.

[29] Jérôme Euzenat and Petko Valtchev. Similarity-based ontology alignment in owl-lite. In *ECAI*, pages 333–337, 2004.

[30] Jérôme Euzenat, David Loup, Mohamed Touzani, and Petko Valtchev. Ontology alignment with ola. In *In Proceedings of the 3rd EON Workshop, 3rd International Semantic Web Conference*, pages 59–68, 2004.

[31] Jean-Rémy Falleri, Marianne Huchard, Mathieu Lafourcade, and Clémentine Nebut. Metamodel matching for automatic model transformation generation. In *MoDELS '08*.

[32] Musen MA Ghazvinian A, Noy NF. Creating mappings for ontologies in biomedicine: simple methods work. In *AMIA*, 2009.

[33] Fausto Giunchiglia and Pavel Shvaiko et al. S-match: an algorithm and an implementation of semantic matching. In *In Proceedings of ESWS*, pages 61–75, 2004.

[34] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, pages 907–928, 1995.

[35] Ramanathan V. Guha, Rob McCool, and Eric Miller. Semantic search. In *WWW*, pages 700–709, 2003.

[36] Alon Y. Halevy. Why your data won't mix: Semantic heterogeneity. *ACM Queue*, pages 50–58, 2005.

[37] Fayçal Hamdi, Brigitte Safar, Nobal B. Niraula, and Chantal Reynaud. Taxomap alignment and refinement modules: results for oaei 2010. In *OM*, 2010.

[38] Md. Seddiqui Hanif and Masaki Aono. An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *J. Web Sem.*, pages 344–356, 2009.

[39] Wei Hu, Jianfeng Chen, Gong Cheng, and Yuzhong Qu. Objectcoref & falcon-ao: results for oaei 2010. In *OM*, 2010.

[40] Wei Hu, Yuzhong Qu, and Gong Cheng. Matching large ontologies: A divide-and-conquer approach. *Data Knowl. Eng.*, pages 140–160, 2008.

[41] Jakob Huber, Timo Sztyler, Jan Nößner, and Christian Meilicke. Codi: Combinatorial optimization for data integration: results for oaei 2011. In *OM*, 2011.

[42] Yves R. Jean-Mary and Mansur R. Kabuka. Asmov: Results for oaei 2008. In *OM*, 2008.

[43] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology matching with semantic verification. pages 235–251, 2009.

[44] 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*, pages 281–295. 2011.

[45] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In *ISWC*, pages 273–288, 2011.

[46] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Yujiao Zhou, and Ian Horrocks. Large-scale interactive ontology matching: Algorithms and implementation. In *ECAI*, pages 444–449, 2012.

[47] Ogden C. K. and Richards I. A. The meaning of meaning. In *Harvest HBJ*, 1989.

[48] Marouen Kachroudi, Essia Ben Moussa, Sami Zghal, and Sadok Ben Yahia. Ldoa results for oaei 2011. In *OM*, 2011.

[49] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *Knowl. Eng. Rev.*, pages 1–31, 2003.

[50] M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In *IJCAI'01*, 2001.

[51] Michel Klein. Combining and relating ontologies: An analysis of problems and solutions, 2001.

[52] Gerald Kowalski. *Information Retrieval Systems: Theory and Implementation.* Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1997.

[53] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. A description logic primer. *CoRR*, 2012.

[54] Andreas Langegger, Wolfram Wöß, and Martin Blöchl. A semantic web middleware for virtual data integration on the web. In *ESWC*, pages 493–507, 2008.

[55] 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.

[56] Bach Thanh Le, Rose Dieng-Kuntz, and Fabien Gandon. On ontology matching problems. In *ICEIS (4)*, pages 236–243, 2004.

[57] Daniel Lemire, Owen Kaser, and Kamel Aouiche. Sorting improves word-aligned bitmap indexes. *Data Knowl. Eng.*, pages 3–28, 2010.

[58] 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.

[59] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.

[60] 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.

[61] Christoph Mangold. A survey and classification of semantic search approaches. *Int. J. Metadata Semant. Ontologies*, pages 23–34, 2007.

[62] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, New York, 2008.

[63] Ming Mao, Yefei Peng, and Michael Spring. A profile propagation and information retrieval based ontology mapping approach. In *SKG*, pages 164–169, 2007.

[64] Ming Mao, Yefei Peng, and Michael Spring. An adaptive ontology mapping approach with neural network based constraint satisfaction. *J. Web Sem.*, pages 14–25, 2010.

[65] Diana Maynard and Sophia Ananiadou. Term extraction using a similarity-based approach. In *In Recent Advances in Computational Terminology. John Benjamins*, pages 261–278, 1999.

[66] Christian Meilicke. Alignment incoherence in ontology matching phd. thesis., 2011.

[67] 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.

[68] Prasenjit Mitra, Natasha F. Noy, and Anuj Rattan Jaiswal. Omen: a probabilistic ontology mapping tool. In *Proceedings of the 4th international conference on The Semantic Web*, ISWC'05, 2005.

[69] Alvaro E. Monge and Charles Elkan. The field matching problem: Algorithms and applications. In *KDD*, pages 267–270, 1996.

[70] Miklos Nagy, Maria Vargas-Vera, and Piotr Stolarski. Dssim results for oaei 2009. In *OM*, 2009.

[71] Natalya F Noy and Michel C. A. Klein. Ontology evolution: Not the same as schema evolution. *Knowl. Inf. Syst.*, pages 428–440, 2004.

[72] Natalya F. Noy and Deborah L. Mcguinness. Ontology development 101: A guide to creating your first ontology. Technical report, 2001.

[73] Natalya Fridman Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

[74] Natalya Fridman Noy and Mark A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *AAAI/IAAI*, pages 450–455, 2000.

[75] Rong Pan, Zhongli Ding, Yang Yu, and Yun Peng. A bayesian network approach to ontology mapping. In *ISWC*, ISWC'05, pages 563–577, 2005.

[76] Shvaiko Pavel and Jerome Euzenat. Ontology matching: State of the art and future challenges. *IEEE TKDE*, 2011.

[77] 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.

[78] Catia Pesquita, Cosmin Stroe, Isabel F. Cruz, and Francisco M. Couto. Blooms on agreementmaker: results for oaei 2010. In *OM*, 2010.

[79] M. F. Porter. An algorithm for suffix stripping. pages 313–316. 1997.

[80] Yuzhong Qu, Wei Hu, and Gong Cheng. Constructing virtual documents for ontology matching. In *WWW*, pages 23–31, 2006.

[81] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, pages 334–350, 2001.

[82] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, 1995.

[83] Quentin Reul and Jeff Z. Pan. Kosimap: Ontology alignments results for oaei 2009, 2009.

[84] Frederik C. Schadd and Nico Roos. Maasmatch results for oaei 2011. In *OM*, 2011.

[85] N. Seco, T. Veale, and J. Hayes. An intrinsic information content metric for semantic similarity in WordNet. *Proc. of ECAI*, pages 1089?1090–1089?1090, 2004.

[86] C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, pages 50–64, 1951.

[87] T. Smith and M. Waterman. *Journal of Molecular Biology*, pages 195–197, 1981.

[88] David Sánchez, Montserrat Batet, and David Isern. Ontology-based information content computation. *KBS*, pages 297–303, 2011.

[89] 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.

[90] Giorgos Stoilos, Giorgos B. Stamou, and Stefanos D. Kollias. A string metric for ontology alignment. In *ISWC Conference*, pages 624–637, 2005.

[91] Ljiljana Stojanovic. *Methods and tools for ontology evolution.* PhD thesis, 2004.

[92] William Sunna and Isabel F. Cruz. Structure-based methods to enhance geospatial ontology alignment. In *GeoS*, pages 82–97. Springer, 2007.

[93] Vojtech Svátek and Petr Berka. Ontofarm: Towards an experimental collection of parallel ontologies. In *In: Poster Session at ISWC*, 2005.

[94] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning object identification rules for information integration. pages 607–633, 2001.

[95] Amos Tversky. Features of similarity. *Psychological Review*, 84:327–352, 1977.

[96] P.R.S. Visser, Pepijn R. S. Visser, Dean M. Jones, T. J. M. Bench-capon, and M. J. R. Shave. An analysis of ontology mismatches; heterogeneity versus interoperability, 1997.

[97] Peng Wang. Lily results on seals platform for oaei 2011. In *OM*, 2011.

[98] Peng Wang and Baowen Xu. Lily: Ontology alignment results for oaei 2009. In *OM*, 2009.

[99] William E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.

[100] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* 1999.