M.Sc. Engg. Thesis

# Ontology Matching by Applying Parallelization and Distribution of Matching Task within Clustering Environment

By
Tanni Mittra

Submitted to
Department of Computer Science and Engineering
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1205

September 15, 2015

The thesis titled "**Ontology Matching by Applying Parallelization and Distribution of Matching Task within Clustering Environment**", submitted by Tanni Mittra, Roll No. 0412052001, Session April 2012, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on September 15, 2015.

## Board of Examiners

1. _____

Dr. Muhammad Masroor Ali — Chairman
Professor — (Supervisor)
Department of CSE
BUET, Dhaka 1205.

2. _____

Head — Member
Department of CSE — (Ex-Officio)
BUET, Dhaka 1205.

3. _____

Dr. M. Kaykobad — Member
Professor
Department of CSE
BUET, Dhaka 1205.

4. _____

Dr. Md. Abul Kashem Mia — Member
Professor
Department of CSE
BUET, Dhaka 1205.

5. _____

Dr. Chowdhury Mofizur Rahman — Member
Professor — (External)
Department of CSE
United International University, Dhaka 1209, Bangladesh.

# Candidate's Declaration

This is to certify that the work presented in this thesis entitled "**Ontology Matching by Applying Parallelization and Distribution of Matching Task within Clustering Environment**" is the outcome of the investigation carried out by me under the supervision of Professor Dr. Muhammad Masroor Ali in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka. It is also declared that neither this thesis nor any part thereof has been submitted or is being currently submitted anywhere else for the award of any degree or diploma.

Tanni Mittra
Candidate

# Contents

# List of Figures

# List of Tables

# Acknowledgements

First of all, I would like to declare that all the appraisals belong to the Almighty **GOD**.

I would like to express my deep gratitude to my supervisor Professor Dr. Muhammad Masroor Ali for introducing me to the fascinating and prospective field of semantic web and ontology matching. I have learned from him how to carry on a research work, how to write, speak and present well. I thank him for his patience in reviewing so many drafts with inaccuracies, errors and for correcting my proofs and language, suggesting new ways of thinking, leading to the right way, and encouraging me to continue my research work. I again express my indebtedness, sincere gratitude and profound respect to him for his continuous guidance, suggestions and whole hearted supervision throughout the progress of this work.

I convey my heartfelt reverence to my parents and other family members for giving their best support throughout my work to overcome the tedium of repetitive trials to new findings.

Finally, every honor and every victory on earth is due to GOD, descended from Him and must be ascribed to Him. He has endowed me with good health and with the capability to complete this work. I deeply express my sincere gratitude to the endless kindness of GOD.

# Abstract

Recent advances in information and communication technology make huge amount of heterogeneous information available for us. But semantically integration of information and provide machine understandable meaning to information is still a great challenge in current web technology. In overcoming the challenges, ontology matching which is introduced by semantic web technology plays a vital role. In this thesis, we propose a new method of ontology matching using parallelization and distribution technique. To apply parallelism, we develop a partitioning algorithm by using property-by-class and subclass of relationship, which partitions the ontology into smaller clusters. Then the clusters from different ontologies are matched based on terminological and structural similarity with semantic verification. All these tasks of matching are handled in a parallel way and all the tasks are distributed over the computational resources. Thus, we significantly reduce the time complexity and space complexity of large scale matching task. Our proposed method reduces misaligned pairs while increasing correct aligned concepts. Validity of our claims have been substantiated through different experiments on small and large ontologies.

# Chapter 1

# Introduction

## 1.1 Ontology Matching

Semantic web [2–4] is an advanced concept that has made revolutionary changes in current web technology of World Wide Web. It is the extension of current web and is built on top of it. One common reason behind the concept of semantic web is to introduce a smart data integration agent. In current web technology, data integration is not smartly handled. For example, if we search in the web for some restaurant for particular kind of food, first, we have to find out some restaurant which serves that food and then search in Google map for the location. We do this kind of information or data integration on the web on a daily basis, but now we hope that a machine will do the task for us. Another common problem of current web is that, the machine cannot understand what it represents. Suppose we want to search information about configuration of DSLR camera. But when we search for the term by writing DSLR, we find many unusual web pages containing the term DSLR. Because our machine does not understand what it represents. The only thing it understands is HTML tag. Besides telling a web browser about how to present my Web page, these HTML constructs do not convey any useful information about the underlying resource. Therefore, other than these HTLM tokens [5] an web page would simply represent a string of characters that look no different from any other web document. To solve these problems semantic web provides machine-understandable meaning and a standard way of data integration to the current web. As a result computers can understand the web documents and therefore can automatically accomplish tasks that have

been otherwise conducted manually on a large scale. This task of semantic web is done by constructing a global vocabulary for web pages that have meaning coded inside its term.

A web page in the semantic web markup the vocabulary file. This file is indeed quite special; its existence has turned the crawler into a smart agent. When the crawler reaches a webpage, it follows the link on it to locate this markup file and furthermore, it is able to understand the meaning of a particular word from the markup document. The markup vocabulary document contanis the semantic of a particular word and thus it will get the meaning of the terms used in the vocabulary. This vocabulary is called ontology and it is specific to domains that may be a specific subject or area such as agriculture, anatomy, education, food etc. Ontology encompasses several data or conceptual models for example, classifications, database schemas, fully axiomatized theories. Ontologies tend to be put everywhere. They are viewed as the silver bullet for many applications, such as information integration, peer-to-peer systems, electronic commerce, semantic web services, social networks and so on. They indeed are a practical means to conceptualize what is expressed in a computer format [6] .

However, in open or evolving systems such as the semantic web different parties would in general adopt different ontologies. Thus, just using ontologies, like just using XML, does not reduce heterogeneity. It raises heterogeneity problems at a higher level. Since the domains have their own data vocabulary, the growth of disparate domain knowledge may cause data redundancy. To reduce data redundancy we have to manage semantic heterogeneity among them. Among the various solutions of heterogeneity problem, ontology matching [7, 8] has taken a critical place. Ontology alignment or matching is the process of determining correspondences between concepts that are semantically equivalent. It is an important operation in many traditional applications like ontology merging, query answering, data translation, data warehouses etc. [7].

The evaluation in the recent years indicates that ontology matching has made a measurable improvement [7]. Considerable work has been done on automating the process of ontology alignment. But still there are some challenges to improve the performance of ontology matching. Among them, increasing the efficiency of matching task is still a great challenge. Because, once quality is achieved, increasing the efficiency of matching task is of prime importance.

To our best knowledge, current matching tools are not optimized for resource consumption and efficiency handling. So overall, the challenge is to come up with scalable ontology matching

reference solutions. To overcome this challenge our proposed method has two major criteria. Firstly, it reduces execution time by performing the matching task by applying parallelization and distribution techniques in a clustering environment. Secondly, it performs wider automation in acquisition of reference alignments of large scale data among several ontologies. The approach presented in this thesis uses cluster computing mechanism to introduce parallelization and distribution of matching tasks with available computational resources. It uses property-class relationship with subclassof relations to cluster ontology. As a result, all the cardinality relationships and instances are preserved. The issue of pairwise comparison complexity is reduced by applying only a simple version of the linguistic and structure based strategy with semantic verification. In the experimental section we compare our approach with some well established methods and with some widely used dataset for different experiments.

## 1.2 Objectives with Specific Aims

The objectives of this thesis are as follows:

- To perform wider automation in acquisition of reference alignments of large scale data among several ontologies.
- Large scale ontology size leads to construction of inefficient matcher. The problem of improving the matching's efficiency and scalability is addressed.
- Cluster computing mechanism is incorporated to introduce parallelization and distribution of matching tasks with available computational resources
- Pair-wise comparisons increase complexity of matching task. The issue of complexity is handled by applying only a simple version of the linguistic based strategies.
- An ontology matcher is designed that can be deployed in real world application to get benefit of running in high-performance computing environments and using full potential of computing resources.

## 1.3 Summary of Contributions

The main contributions of the thesis are enumerated below:

- An efficient and scalable ontology matcher capable of matching ontology in high-performance computing environments.
- A matcher capable of matching the input ontology with different ontologies at the same time.
- A comparative study of our proposed algorithm with the representative state-of-the-art algorithms.
- A matcher capable of finding out semantic similarity with terminological and structural similarity.

## 1.4 Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 presents previous work of ontology matching. Chapter 3 defines the terminologies and notations used in this thesis. Chapter 4 presents the computation of structural proximities, the partitioning algorithm and the construction of clusters with different alignment approach. Chapter 5 reports experimental results on both synthetic and real world data sets. Finally, Chapter 6 enumerates the attainments of this thesis and then concludes the thesis suggesting the future extensions possible.

# Chapter 2

# Related Work

In recent years, different types of ontology matching technique is determined by different researchers. A lot of work has addressed the ontology matching problem but most of them solve the matching problem of small ontology and there are few generic approaches that raise the issue of matching large ontologies [8]. To our best knowledge, very few ontology matcher can handle both large and small ontology. Among the several dozens of systems that have appeared in these recent years, we selected some which have repeatedly participated to the Ontology Alignment Evaluation Initiative (OAEI) campaigns in order to have a basis for comparisons. Below, we are the discussion of these systems in more details.

## 2.1 Large Ontology Matching

To match large ontologies, a few light-weight ontology matching approaches can be directly used. For example, edit-distance based methods [9] compute the similarity between two strings (e.g. rdfs:label(s)) by counting the number of operations required to transform one of them into the other. Because they only compare two strings in each time, they do not have the scalability limit and can be applied to match ontologies with any size. On the other hand, some work adapts the matching algorithms to reduce memory consumption. For example, the work in [10] simplifies a graph matching algorithm to match two large medical taxonomies by only considering the direct children and grandchildren of entities. The major ontology matching systems with the relevant technologies are described below.

### SAMBO

SAMBO is a system for matching and merging biomedical ontologies [11]. It handles ontologies in OWL and outputs 1:1 alignments between concepts and relations. The system uses various similarity based matchers, including terminological similarity where n-gram, edit distance method are used. The results of these matchers are combined via a weighted sum with pre-defined weights; structural, through an iterative algorithm that checks if two concepts occur in similar positions with respect to is-a or part-of hierarchies relative to already matched concepts, with the intuition that the concepts under consideration are likely to be similar as well.

The results produced by these matchers are combined based on user-defined weights. Then, filtering based on thresholds is applied to come up with an alignment suggestion, which is further displayed to the user for feedback (approval, rejection or modification). Once matching has been accomplished, the system can merge the matched ontologies, compute the consequences, check the newly created ontology for consistency, etc. SAMBO has been subsequently extended into a toolkit for evaluation of ontology matching strategies, called KitAMO.

### FALKON

Falcon is an automatic divide-and-conquer approach to ontology matching [12]. It handles ontologies in RDFS and OWL. It has been designed with the goal of dealing with large ontologies (of thousands of entities). The approach operates in three phases: (i) partitioning ontologies, (ii) matching blocks, and (iii) discovering alignments. The first phase starts with a structure-based partitioning to separate entities (classes and properties) of each ontology into a set of small clusters. Partitioning is based on structural proximities between classes and properties e.g. how closely are the classes in the hierarchies of rdfs:subClassOf relations and on an extension of the Rock agglomerative clustering algorithm . Then it constructs blocks out of these clusters. In the second phase the blocks from distinct ontologies are matched based on anchors (pairs of entities matched in advance), i.e., the more anchors are found between two blocks, the more similar the blocks are. In turn, the anchors are discovered by matching entities with the help of the I-SUB string comparison technique . The block pairs with high similarities are selected based on a cutoff threshold. Notice that each block is just a small fragment of an ontology. Finally, in

the third phase the results of the so-called V-Doc (a linguistic matcher) and GMO (an iterative structural matcher) techniques are combined via sequential composition to discover alignments between the matched block pairs. Ultimately, the output alignment is extracted through a greedy selection

**DSSim**

DSSim is an agent-based ontology matching framework. The system handles large-scale ontologies in OWL and SKOS and computes 1:1 alignments with equivalence and subsumption relations between concepts and properties. It uses the Dempster-Shafer [13] theory in the context of query answering . Specifically, each agent builds a belief for the correctness of a particular correspondence hypothesis. Then, these beliefs are combined into a single more coherent view in order to improve correspondence quality. The ontologies are initially partitioned into fragments. Each concept or property of a first ontology fragment is viewed as a query, which is expanded based on hypernyms from WordNet , viewed as background knowledge. These hypernyms are used as variables in the hypothesis to enhance the beliefs. The expanded concepts and properties are matched syntactically to the similar concepts and properties of the second ontology in order to identify a relevant graph fragment of the second ontology. Then, the query graph of the first ontology is matched against the relevant graph fragment of the second ontology. For that purpose, various terminological similarity measures are used, such as Monger-Elkan and Jaccard distances, which are combined using Dempster's rule. Similarities are viewed as different experts in the evidence theory and are used to assess quantitative similarity values (converted into belief mass functions) that populate the similarity matrices. The resulting correspondences are selected based on the highest belief function over the combined evidences. Eventual conflicts among beliefs are resolved by using a fuzzy voting approach equipped with four ad hoc if-then rules. The system does not have a dedicated user interface but uses that of the AQUA question answering system able to handle natural language queries.

**RiMOM**

RiMOM is a dynamic multi-strategy ontology matching framework [14]. It extends a previous version of the system that focused on combining multiple matching strategies, through risk minimization of Bayesian decision. The new version quantitatively estimates the similarity

characteristics for each matching task. These characteristics are used for dynamicly selecting and combining the multiple matching methods. Two basic matching methods are employed: (i) linguistic similarity (edit distance over entity labels, vector distance among comments and instances of entities) and (ii) structural similarity (a variation of Similarity Flooding implemented as three similarity propagation strategies: concept-to-concept, propertyto- property and concept-to-property). In turn, the strategy selection uses label and structure similarity factors, obtained as a preprocessing of the ontologies to be matched, in order to determine what information should be employed in the matching process. Specifically, the strategy selection dynamically regulates the concrete feature selection for linguistic matching, the combination of weights for similarity combination, and the choice of the concrete similarity propagation strategy. After similarity propagation, the matching process concludes with alignment refinement and extraction of the final result.

**ASMOV**

ASMOV (Automatic Semantic Matching of Ontologies with Verification) is an automatic approach for ontology matching that targets information integration for bioinformatics [1]. Overall, the approach can be summarized in two steps: (i) similarity calculation, and (ii) semantic verification. It takes as input two OWL ontologies and an optional input alignment and returns as output an $n:m$ alignment between ontology entities (classes and properties). In the first step it uses lexical (string equality, a variation of Levenshtein distance), structural (weighted sum of the domain and range similarities) and extensional matchers to iteratively compute similarity measures between two ontologies, which are then aggregated into a single one as a weighted average. It also uses several sources of general and domain specific background knowledge, such as WordNet and UMLS, to provide more evidence for similarity computation. Then, it derives an alignment and checks it for inconsistency. Consistency checking is pattern based, i.e., that instead of using a complete solver, the system recognizes sets of correspondences that are proved to lead to an inconsistency. The semantic verification process examines five types of patterns, e.g. disjoint-subsumption contradiction, subsumption incompleteness. This matching process is repeated with the obtained alignment as input until no new correspondences are found.

**Anchor-Flood**

The Anchor-Flood approach aims at handling efficiently particularly large ontologies [15]. It inputs ontologies in RDFS and OWL and outputs 1:1 alignments. The system starts with a pair of similar concepts from two ontologies called an anchor, e.g. all exactly matched normalized concepts are considered as anchors. Then, it gradually proceeds by analyzing the neighbors, i.e., super-concepts, sub-concepts, siblings, of each anchor, thereby building small segments (fragments) out of the ontologies to be matched. The size of the segments is defined dynamically starting from an anchor and exploring the neighboring concepts until either all the collected concepts are explored or no new matching pairs are found. The system focuses on (local) segment-to-segment comparisons, thus it does not consider the entire ontologies which improves the system scalability. It outputs a set of correspondences between concepts and properties of the semantically connected segments. For determining the correspondences between segments the approach relies on terminological (WordNet and Winkler-based string metrics) and structural similarity measures, which are further aggregated by also considering probable misalignments. The similarity between two concepts is determined by the ratio of the number of terminologically similar direct super-concepts on the number of total direct super-concepts. Retrieved (local) matching pairs are considered as anchors for further processing. The process is repeated until there are no more matching pairs to be processed.

**AgreementMaker**

AgreementMaker is a system comprising a wide range of automatic matchers, an extensible and modular architecture, a multi-purpose user interface, a set of evaluation strategies, and various manual, e.g. visual comparison, and semi-automatic features, e.g. user feedback [13]. It has been designed to handle largescale ontologies based on the requirements coming from various domains, such as the geospatial and biomedical domains. The system handles ontologies in XML, RDFS, OWL, N3 and outputs 1:1, $1:m$, $n:1$, $n:m$ alignments. In general, the matching process is organized into two modules: similarity computation and alignment selection. The matchers of the first layer compare concept features, such as labels, comments, instances, which are represented as TFIDF vectors used with a cosine similarity metric. The second layer uses structural ontology properties and includes two matchers called descendants similarity inheri-

tance (if two nodes are matched with high similarity, then the similarity between the descendants of those nodes should increase) and siblings similarity contribution. At the third layer, a linear weighted combination is computed over the results coming from the first two layers, whose results are further pruned.

## 2.2 Matching Large Ontology in Particular Area

In addition, there exist some domain-specific approaches that address matching large ontologies in particular areas such as anatomy, biology and geography. [16, 17] uses biology and geography ontology to perform matching operation. Usually, they utilize certain light-weight methods to gain initial candidates and use domain knowledge to infer alignments. AOAS [16] is a representative tool that is heavily specialized on matching biomedical ontologies and makes extensive use of medical background knowledge. For the Anatomy track in OAEI 2007, it takes a broader ontology FMA as the background knowledge to further improve the initial results. Because these approaches heavily depend on domain knowledge, they are not general-purpose solutions and easily fail when such knowledge is unavailable. But, if there is proper background knowledge, they can find some complex and interesting alignments

## 2.3 Matching Large Ontology using Parallelization

Works mentioned in [18] and [19] employ some parallelization and distribution techniques for ontology matching. However, we argue that our proposed method is superior to these aforementioned works. As discussed in [18], they used life science ontologies to determine ontology mappings. For matching, they have used inter match and intra matcher parallelization technique. In intra matcher technique, they used the complete ontology for matching each time, which required more memory. In inter matcher technique, they used size-based ontology partitioning i.e. partition whole ontology into fixed numbers of partitions. They have used internal decomposition of individual matchers and for each matcher, they have partitioned the whole ontology. The overall techniques used in the aforementioned paper is time consuming and the degree of parallelism is limited, because, the result of one matcher depends on previously executed matcher. However, our proposed method partitions the input ontology only once and

never use size based partitioning. It handles partitioning of ontology by maintaining neighboring relationship between entities. In [19], they only partitioned the input ontology into several parts, where each part of the portioned ontology is matched with each concept of another compared ontology, in a distributed and parallel manner. But our proposed method partitions both ontologies into clusters and matching is performed between clusters of two ontologies. Moreover our proposed method uses semantic verification technique [1], which is not used in the above mentioned methods.

There are many possible ways to perform partitioning and applying parallelism but finding the most effective approach is still an open research problem [20]. So by considering this, we proposed a matching techniques in [21], where we developed a partitioning algorithm by using relationships between properties and classes. We have used parallelism technique to find out terminological and structural similarity with semantic verification between ontologies. In our previous work [21], semantic verification and lexical similarity calculation was not effectively handled. As semantic similarity plays vital role in ontology matching, so we have extended our previous work by adding additional techniques of semantic verification and by including lexical similarity calculation to make the matching task more robust. Again, we have conducted our experiment on real word dataset as well as very large datasets to evaluate the efficiency of the proposed method.

## 2.4 Analytical Summary

The following can be observed concerning the considered systems:

- The approaches equally pursue the development of generic matchers, e.g. Falcon, RiMOM, Anchor-Flood, as well as those focusing on particular application domains, e.g. SAMBO, ASMOV, that target primarily biomedical applications.
- Most of the systems under consideration declare to be able to handle efficiently large-scale ontologies.
- Although all systems can deal with OWL (being an OAEI requirement), many of them can be applied to RDFS or SKOS.

- Most of the systems focus on discovering 1:1 alignments, but yet several systems are able to discover $n : m$ alignments. Moreover, most of the systems focus on computing equivalence relations, with the exception of DSSim, which is also able to compute subsumption relations.

- Many systems are not equipped with a graphical user interface, with several exceptions, such as SAMBO, DSSim, and AgreementMaker.

- Semantic and extensional methods are still rarely employed by the matching systems. In fact, most of the approaches are quite often based only on terminological and structural methods.

- Many systems have focussed on combining and extending the known methods. For example, the most popular of these are variations of edit distance and WordNet matchers as well as iterative similarity propagation as adaptation of the Similarity Flooding algorithm. Thus, the focus was not on inventing fundamentally new methods, but rather on adapting and extending the existing

# Chapter 3

# Preliminaries

## 3.1 Basic Terminology

An ontology is a formal, explicit specification of a shared conceptualization [22]. In this thesis, we use a basic definition of ontologies based on the RDF graph model, which is general enough to cover most of the state of the art large ontologies. In this section we define some basic terminologies regarding ontology.

### 3.1.1 Semantic Web

The Semantic Web is a collaborative movement led by international standards body, the World Wide Web Consortium (W3C). According to the W3C, The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. It is constructed by linking current Web pages to a structured data set that indicates the semantics of this linked page. A smart agent, which is able to understand this structure data set, will then be able to conduct intelligent actions and make educated decisions on a global scale. In the traditional Web environment, each Web page only provides information for computers to display the page, not to understand it. The page is solely intended for human eyes. While in semantic web we can construct a vocabulary set that contains the important words(concepts, classes) for a given domain, and the semantics and knowledge are coded into this set; more importantly, this

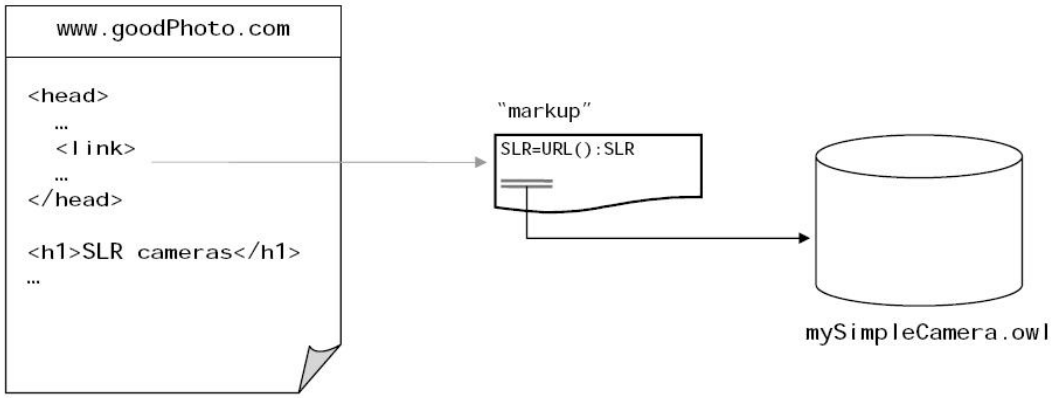


Figure 3.1: A Web Page in the Semantic Web Environment.

set has to be constructed using some structured data. We then markup a web page by adding a pointer in its metadata section. This pointer points to the appropriate vocabulary set, this is how we add semantics to each page. Fig. 3.1 represents how a webpage link to the vocabulary.

### 3.1.2 Resource Description Framework

RDF is the basic building block for supporting the Semantic Web. RDF is to the Semantic Web what HTML has been to the Web. RDF is a language recommended by W3C [3]. It provides a basis for coding, exchanging and reusing structured metadata. RDF is structured i.e. it is machine-understandable. Machines can do useful operations with the knowledge expressed in RDF. RDF allows interoperability among applications exchanging machine understandable information on the Web. The basic element of RDF is resource, property. A resource therefore is anything that is being described by RDF expressions. It can be a Web page, part of a Web page (a word on a page, for instance), the whole Web site, or even a real-world object, such as a book, a human being, a dog, it can be anything. Property is a resource that has a name and can be used as a property i.e. it can be used to describe some specific aspect, characteristic, attribute, or relation of the given resource.

### 3.1.3 Resource Description Framework Schema

RDFS is a language one can use to create a vocabulary for describing classes, subclasses, and properties of RDF resources. It is a recommendation from W3C. The RDFS language also associates the properties with the classes it defines. RDFS can add semantics to RDF predicates

and resources: it defines the meaning of a given term by specifying its properties and what kinds of objects can be the values of these properties. RDFS is written in RDF. In fact, not only is RDFS written in RDF, but RDFS also uses the same data model as RDF i.e. a graph or triples. In this sense, RDFS can be viewed as an extension of RDF.

### 3.1.4 Web Ontology Language

OWL (Web Ontology Language) is the latest recommendation of W3C and is probably the most popular language for creating ontologies today. OWL = RDF schema + new constructs for expressiveness. OWL and RDF schema have the same purpose: to define classes, properties, and their relationships. However, compared to RDF schema, OWL gives us the capability to express much more complex and richer relationships.

### 3.1.5 Classes

Classes are concepts that are also called type, sort, category and kind – can be defined as an extension or an intension. According to an extensional definition, they are abstract groups, sets, or collections of objects. According to an intensional definition, they are abstract objects that are defined by values of aspects that are constraints for being member of the class. Classes may classify individuals, other classes, or a combination of both. Some examples of classes are:

- Person, the class of all people, or the abstract object that can be described by the criteria for being a person,
- Vehicle, the class of all vehicles, or the abstract object that can be described by the criteria for being a vehicle and
- Car, the class of all cars, or the abstract object that can be described by the criteria for being a car

The classes of an ontology may be extensional or intensional in nature. A class is extensional if and only if it is characterized solely by its membership. More precisely, a class C is extensional if and only if for any class $C_1$, if $C_1$ has exactly the same members as C, then C and $C_1$ are identical. If a class does not satisfy this condition, then it is intensional. While

extensional classes are more well-behaved and well-understood mathematically, as well as less problematic philosophically, they do not permit the fine grained distinctions that ontologies often need to make. For example, an ontology may want to distinguish between the class of all creatures with a kidney and the class of all creatures with a heart, even if these classes happen to have exactly the same members. In most upper ontologies, the classes are defined intensionally. Intensionally defined classes usually have necessary conditions associated with membership in each class. Some classes may also have sufficient conditions, and in those cases the combination of necessary and sufficient conditions make that class a fully defined class.

### 3.1.6 Individuals

An individual is minimally introduced by declaring it to be a member of a class. Fig. 3.2 shows an example of individuals.

here rdf:type is an RDF property that ties an individual to a class of which it is a member. CentralCoastRegion (a specific area) is member of Region, the class containing all geographical regions. So in that case CentralCoastRegion is an intance of class Region.

### 3.1.7 Properties

A property is a binary relation. Two types of properties are distinguished.

- Datatype properties, relations between instances of classes and RDF literals and XML Schema datatypes.
- Object properties, relations between instances of two classes. Note that the name object property is not intended to reflect a connection with the RDF term rdf:object.

  When a property is defined, there are a number of ways to restrict the relation. The domain and range can be specified. The property can be defined to be subproperty of

```
<owl:Thing rdf:ID="CentralCoastRegion" />
<owl:Thing rdf:about="#CentralCoastRegion">
   <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

Figure 3.2: An Individual Example.

an existing property. It is possible to specify property characteristics, which provides a powerful mechanism for enhanced reasoning about a property such as transitive property, symmetric property, functional property etc.

### 3.1.8 Relationships

Relationships between classes in an ontology specify how classes are related to other class. Typically a relation is of a particular type that specifies in what sense the object is related to the other object in the ontology. Fig. 3.3 shows an example which will help the reader to understand relationship. An important type of relation is the subsumption relation (is-a-superclass-of, the converse of is-a, is-a-subtype-of or is-a-subclass-of). This defines which objects are classified by which class. Fig. 3.3 shows that the class Ford Explorer is-a-subclass-of 4-Wheel Drive Car, which in turn is-a-subclass-of Car.

Another common type of relations is the mereology relation, written as part-of, that represents how objects combine together to form composite objects. For example, if we extended our example ontology to include concepts like Steering Wheel, we would say that a "Steering Wheel is-by-definition-a-part-of-a Ford Explorer" since a steering wheel is always one of the components of a Ford Explorer.

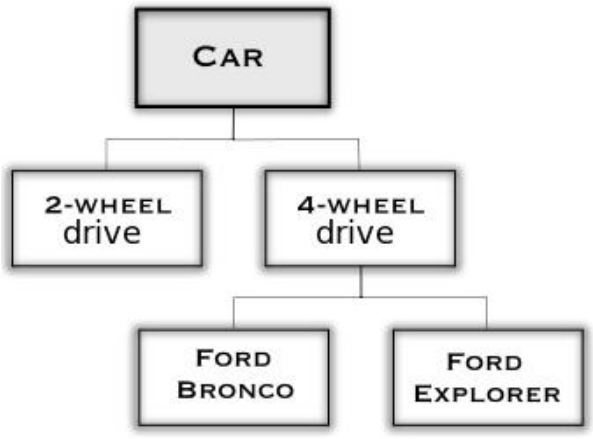


Figure 3.3: Ontology Relationships.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  <xmlns="http://www.yuchen.net/photography/Camera#">
    <SLR rdf:ID="Nikon-D70">
       <weight>1.4 lbs</weight>
    </SLR>
</rdf:RDF>
```

Figure 3.4: An Ontology Example.

### 3.1.9 Ontology

An ontology is used to describe and represent an area of knowledge [8]. In other words, ontology is domain specific; it is not there to represent all knowledge, but an area of knowledge. A domain is simply a specific subject area or sphere of knowledge, such as photography, medicine, real estate, education, etc. Again, an ontology contains terms and the relationships among these terms. Terms are often called classes, or concepts; these words are interchangeable. The relationships between these classes can be expressed by using a hierarchical structure: superclasses represent higher-level concepts and subclasses represent finer concepts, and the finer concepts have all the attributes and features that the higher concepts have. Besides the aforementioned relationships among the classes, there is another level of relationship expressed by using a special group of terms: properties. These property terms describe various features and attributes of the concepts, and they can also be used to associate different classes together. Therefore, the relationships among classes are not only superclass or subclass relationships, but also relationships expressed in terms of properties. Fig. 3.4 represents an example of Ontology. To summarize, an ontology has the following properties:

- It is domain specific.
- It defines a group of terms in the given domain and the relationships among them.

### 3.1.10 Ontology Clustering

Cluster is a set of entities of an ontology. Generally speaking, a cluster can be viewed as a subset of an ontology . A cluster contains a set of entities or individuals with all their property decleration and property restriction relationship. A cluster is comprised of a set of similar

entities grouped together. A partitioning breaks all the entities in an ontology into a set of clusters, satisfying: (1) for any two different clusters, they are disjoint; and (2) the union of all the clusters equals to the entire set of the entities. Let $O_1$ is an ontology and E is a set of entity. Culstering C of $O_1$ breaks E into set of clusters where C = $c_1$, $c_2$, $c_3$, $c_4$..... $c_n$ which satisfied (i) $c_1 \cap c_2 = \phi$ and (ii) $c_1 \cup c_2 \cup c_3 \cap c_4$ = E.

### 3.1.11 Relationship between Properties and Classes

Property is a resource that has a name and can be used as a property; i.e., it can be used to describe some specific aspect, characteristic, attribute, or relation of the given resource. An RDF statement is used to describe properties of resources. It has the following format: resource (subject) + property (predicate) + property value (object). Therefore, in general, an RDF statement indicates that a resource (the subject) is linked to another resource (the object) via an arc labeled by a relation (the property). It can be interpreted as follows: subject has a property predicate, whose value is object. So from the format it is known that property plays an important role in providing relationship between classes. The property value can be a string literal or a resource. Property value provides neighbouring relationship of classes.

## 3.2 Ontology Matching Problem

In this section we first discuss a motivating example and then we provide some basics of ontology matching.

### 3.2.1 Ontology Matching Example

In order to illustrate the matching problem let us use the two simple ontologies $O_1$ and $O_2$ described in Fig. 3.5, where classes are shown in rectangles with rounded corners. In $O_1$, Book is a subclass of class Product, price is an attribute defined on the integer domain and creator is a property. Albert Camus: La chute is a shared instance. Correspondences are shown as thick arrows that link an entity from $O_1$ with an entity from $O_2$. They are annotated with the relation that is expressed by the correspondence: for example, Person in $O_1$ is less general than Human in $O_2$.

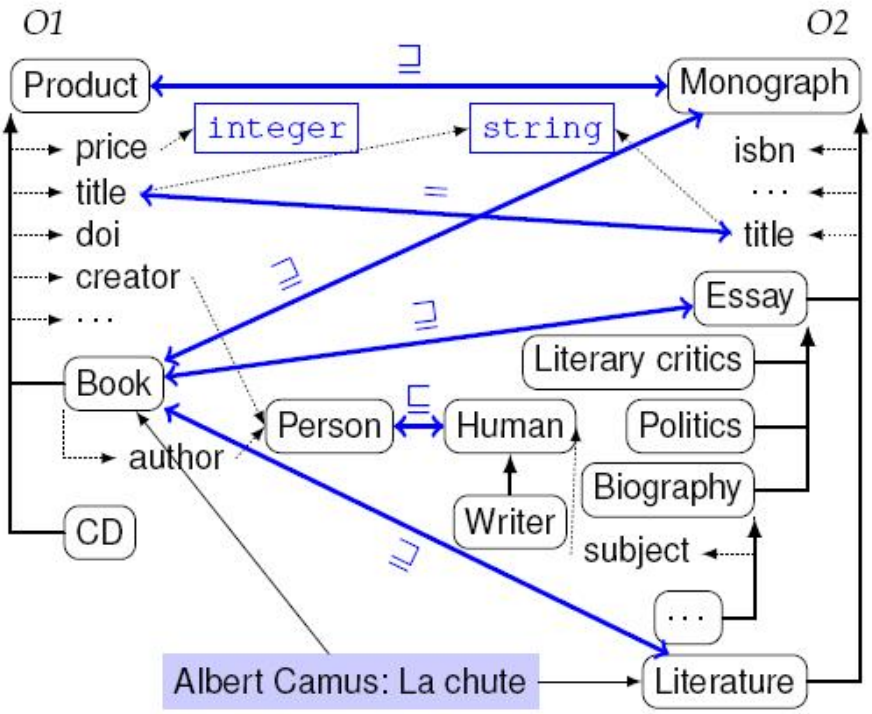


Figure 3.5: Ontology Matching Example (adapted from [1]).

In our example these ontologies contain subsumption statements, property specifications and instance descriptions. The first step in integrating ontologies is matching, which identifies correspondences, namely the candidate entities to be merged or to have subsumption relationships under an integrated ontology. Once the correspondences between two ontologies have been determined they may be used for instance, for generating query expressions that automatically translate instances of these ontologies under an integrated ontology.

### 3.2.2 Ontology Matching

Ontology Alignment is the process of determining correspondence between semantically aligned concepts of different ontology. Alignment is obtained by computing similarity value between pair of classes of ontologies. These alignment may hold one-to-one,one-to-many or many-to many relations between classes. Let $O_1$ and $O_2$ be two ontologies. Matching $O_1$ with $O_2$ finds a set of alignments A= $a_1$, $a_2$, ... $a_n$. Alignment A is defined as a set of correspondences with quadruples [e, f, r, l ] where e and f are the two aligned entities across ontologies, r represents the relation holding between them, and l represents the level of confidence [0, 1].

## 3.3 Present State

Ontology Matching is still one of the hottest topics in the Semantic Web research. Matching ontologies is a critical operation in many application domains, such as Semantic Web, ontology integration, data warehouses, e- commerce, query mediation, etc. It takes two ontologies as input, each consisting of a set of discrete entities and determines the relationships (e.g. equivalence, subsumption) holding between these entities as output. Many diverse solutions to the matching problem have been proposed so far. Specific criteria has been proposed for evaluating and distinguishing between matching approaches. They are evaluated based on 1. the different kind of input of the algorithms, e.g. labels assigned to the entities, entities' internal structure and the type of their attributes, and relations with other entities, 2. the characteristics of the matching process, e.g. approximate or exact nature of its computation, or the interpretation type of the input data e.g. syntactic, external, semantic and 3. the output of the algorithms e.g. whether the form of the matching is a one-to-one, one-to-many or many-tomany correspondence between the ontology entities. Other significant distinctions in the output results reported so far are: a) confidence and probability percentages of the resulted mappings, b) the kind of relations between entities a system can provide (equivalence, subsumption, and incompatibility). Several tools have been developed towards solving the ontology-matching problem, either in a semiautomatic or fully automatic way. Human-involvement during the process is usually in a trade with the precision and recall percentages of the resulted mappings. Fully automated tools are still "looking for" higher accuracy and they are continuously evaluated in international contests e.g. OAEI contest [23]. Still, both automated and semi-automated tools are suffering in their performance. For instance, most of them cannot handle large real-domain ontologies, e.g. Medical[1] or Biology [2], although more and more realistic test-beds are used to evaluate ontology matching tools (scalability problem). Beyond ontology matching methods, tools, and evaluation initiatives/frameworks, recent efforts have been made on ontology-matching-tool-design frameworks [24]. Such frameworks allow developers to use APIs not only for supporting the devising of ontology matching methods but more importantly to provide a mean for synthesizing them towards developing robust tools that produce more accurate mappings. Most recently, ontology matching methods were used to solve the Semantic Web services matchmaking prob-

[1] http://purl.obolibrary.org/obo/go.owl
[2] http://bioportal.bioontology.org/

lem, transforming the problem of discovering web services into a matching problem between an ontology description representing a service request to an ontology description representing an offered service. Although there are a few tools implementing such an approach still there is more to be done towards improving the process.

## 3.4 Challenges

Although a lot have been done towards tackling the problem of ontology matching, the research community still reports open issues that impose new challenges for researchers and underline new directions for the future. The few challenges are described in below subsections:

### 3.4.1 Scalability

Most of the implemented and evaluated ontology matching tools suffer from handling large ontologies (e.g. from Medical, Biology domains). Real problems in specific application contexts require scalable solutions as a first priority. Future ontology matching tools should be able to provide such capability.

### 3.4.2 Speed, Automation, Accuracy Tuning

Tools are currently emphasizing on maximizing specific parameters of their performance such as speed, automation or accuracy. The most common case is the maximization of the performance for one of these parameters, and the other parameters are neglected or largely influenced in a negative manner. Future directions should be towards a fine tuning of all parameters such as the overall performance of an ontology matching tool (and of the application that is integrated in) to be leveraged.

### 3.4.3 Background Knowledge

Experiments of matching an ontology to another while using one (or more) much larger and detailed ontology of the same domain as background knowledge have been recently announced. The extensive use of domain-related background knowledge in the ontology matching process

has positive effects on recall, but does not seem to scale well with large ontologies. The challenge in this case is to adopt such an approach without sacrificing the overall tools' performance. Future directions may be towards discovering an optimum point in the backgroundknowledge/ performance trade-off.

### 3.4.4 Ontology Matching Frameworks

Although ontology-matching-tool-design frameworks have been already developed supporting developers towards not only devising ontology matching methods but also synthesizing them towards realizing new and robust tools that produce more accurate mappings, their performance need to be further investigated. Scalability, speed, and input ontology type compatibility should be further investigated towards delivering a model-framework that could be used by the research community to device ontology matching tools for any user or application preferences.

# Chapter 4

# Methodology

## 4.1 Proposed Methodology

The proposed method of ontology matching is subdivided into four basic steps. After following each of the four steps sequentially, it will produce a set of aligned concepts between two ontologies $O_1$ and $O_2$. The four basic steps are:

- Input Ontology,
- Ontology Clustering,
- Similarity Matching,
- Determine Aligned Entity.

The process flow of Ontology Matching is depicted in Fig. 4.1. Each steps will perform some basic functionality to produce aligned pair between ontologies. Now we will provide a detail description about ontology matching in the following section.

### 4.1.1 Input Ontology

An ontology may be of different size or from different domains. For example, Human ontology contains 3304 entities and it is from the human anatomy domain, Russia1 ontology contains 151 classes and it contains detail description about the content of the travel website. Our proposed

method can take both large or small ontologies as input. After taking an ontology as input, it is forwarded to the next steps to generate aligned concepts between two ontologies.

### 4.1.2 Ontology Clustering

#### Preprocessing

The preprocessing step before clustering creates a helper object model [25] by extracting all the necessary components of ontology and preserving the relationship that exists among them. The main idea behind helper object model is to preserve necessary components of a class like properties, individuals, cardinality and neighboring relationships as a record. Since all attributes and elements of ontology are represented in syntax, so to avoid parsing XML repeatedly, helper model is created for each class by parsing the only once. Thus, for any further operation performed on ontology object model is used instead of parsing full syntax. Fig. 4.2 represents a small part of Russia1 ontology mentioned in [12] and a helper object model of the ontology. This ontology describes the content of a travel web site about Russia.

The helper model is used to calculate structural similarity among internal classes of ontology by using `property-class` and `subclassof` relationship. From the design principle of

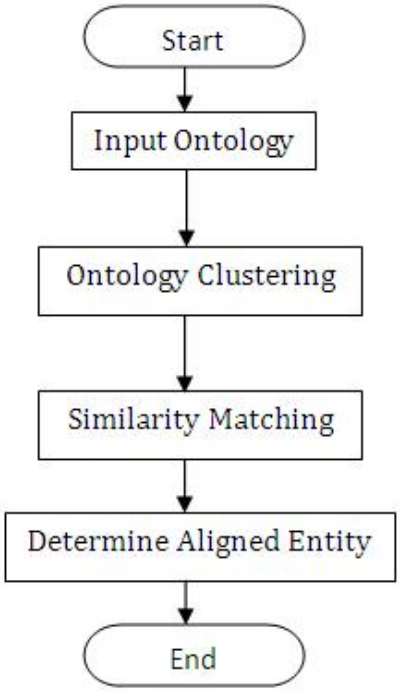


Figure 4.1: Process Flow of Ontology Matching.

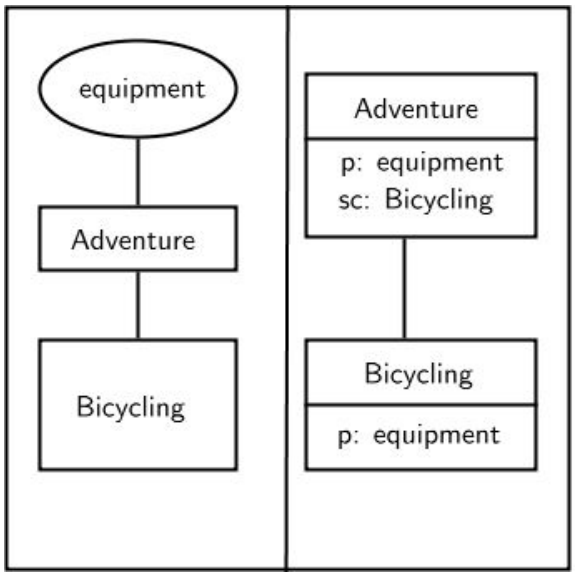


Figure 4.2: Helper Object Model of Russia1 Ontology.

large ontology we come to know that an ontology is designed based on classification and modularization principle. These two issues can be easily solved by `property-class` relationship because the property of an ontology can provide the `subclassof` relationship. Suppose, `Location` is a class of an ontology and `hasPlace` is a common property of `Location` class. As a subclass of `Location` class `Region` and `Space` get the property of `hasPlace`.

Again, ontology property ties an individual to a class and preserves cardinality relationship among entities. Suppose, `Region` is a class and `CentralCoastRegion` is an instance of class `Region`. `rdf:type` is an RDF property that ties an individual to a class. To handle ontologies without property declaration we use hierarchical relation among classes i.e. `subclassof` relationship to find structural similarity.

The steps needed for ontology clustering are discussed below.

### Entity Vector

To perform ontology clustering the first step is to build a property by class matrix. The key formula of building property by class matrix is property-class relationship. The helper object model described in previous section, plays a vital role for determining property-class relationship. As discussed above, this model preserves necessary components of a class like properties, individuals, cardinality and neighboring relationships as a record. From the record set, we can easily find out the number of properties shared by each number of classes including instances

which is the basic component of property by class matrix.

Assume, an ontology $O$ with $|C|$ classes and $|P|$ properties where $C$ represents set of classes and instances and $P$ represents set of properties. The property by class matrix is defined as $(R_{ij})$ where i represents classes and j represents the properties of ontology. The matrix is determined by using the following equation [26],

$$R_{ij} = \begin{cases} 1 & \text{if } C \subseteq P \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

The above equation provides 1 if a class contains a property, otherwise, it provides 0. Fig. 4.3 represents the property-by-class matrix construction of `Book` ontology by following the equation. The `Book` ontology contains five classes $E1$, $E2 \ldots E5$ and three properties $P1$, $P2$ and $P3$. Fig. 4.3 represents that, class $E2$, $E3$, $E4$ contains all three properties so we place 1 in appropriate location and class $E5$ contains only one property $P3$ so we place 1 at location $P3$ and 0 in other locations.

From this property by class matrix, entity vector is calculated. Here, entity vector means vector of the classes of the ontology. As each row of the matrix represents class and each column represents property, so, the entity vectors are stored in the row space of the matrix. As a matter of fact, each row in the property by class matrix is considered as the corresponding entity vector.

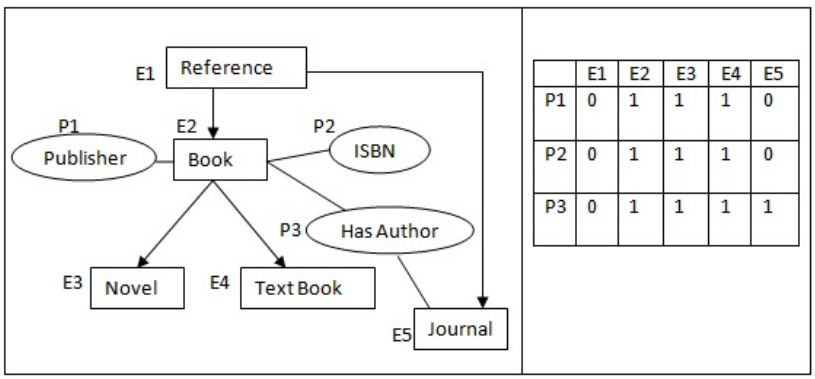


Figure 4.3: Property-by-Class Matrix.

### Internal Structural Similarity Calculation

After finding out the entity vector, the internal structural similarity between two entities is determined by the ratio between intersecting set and union set of their entity vectors. The ratio is equal to zero if there are no intersecting elements between entity vectors and equal to one if all elements intersect. In our proposed method, this ratio is calculated using Jaccard coefficient [27] which is a measurement of asymmetric information on binary variables[1]. This measurement is used for comparing the similarity and diversity between two sets of entities. The following equation is used to calculate Jaccard coefficient [27].

$$Sim(A, B) = \frac{\#(N_C)}{\#(N_A) + \#(N_B) - \#(N_C)} \tag{4.2}$$

where, $A$ and $B$ are binary vectors. $N_A$ is the number of elements in set $A$, $N_B$ is the number of elements in set $B$ and $N_C$ is the number of elements in intersecting set. In our proposed method, the entity vector is represented as binary vector with 1 or 0 representation. By replacing $A$ and $B$ with entity vector $r_i$ and $r_j$, we get the following equation (taken from [26]) to calculate structural similarity.

$$\mathrm{Sim}(r_i, r_j) = \frac{\#(r_i = r_j = 1)}{\#(r_i = 1) + \#(r_j = 1) - \#(r_i = r_j = 1)} \tag{4.3}$$

So, according to the above equation of structural similarity calculation, we can say that two entities are structurally similar if they share same type and almost same number of properties between them.

### Partitioning Ontology

Internal structural similarity discussed above plays an important role in ontology clustering because this similarity provides structural connection i.e. neighboring relationship between classes. In our proposed method, we focus on neighboring relationship of classes to perform clustering of the ontologies. In this section, we have described about how the partitioning algorithm of our proposed method uses internal structural similarity of classes to partition the ontology into smaller clusters.

[1] http://en.wikipedia.org/wiki/Jaccard_index

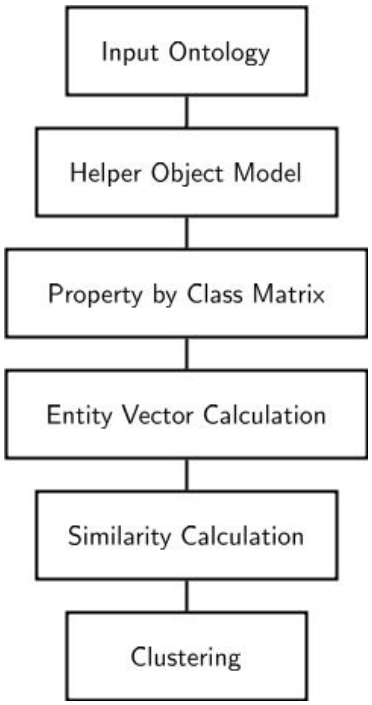


Figure 4.4: Steps of Clustering Algorithm.

Clustering of the ontologies starts with the determination of the seed nodes by computing shared number of properties between classes. Initially, all the classes are the member of the seed node set. If two classes share the same properties, then we remove one class from the seed node set. Otherwise, we check hierarchical relations between them. If they are connected hierarchically, then we again remove one class from the seed node set. We repeat the whole process until each of the classes is checked and thus we get a seed note set. Fig. 4.4 describes the steps used in our proposed partitioning algorithm.

For partitioning classes we begin our work with the remaining classes of ontology except seed classes. First, the $|S|$ seed classes are individually assigned to the $|S|$ clusters as their initial members. Here, each seed node acts as the starting point for each cluster in the cluster set $S$. Then we calculate the internal structural similarity value of the remaining class members with each of the classes in the cluster set. We choose the maximum value among them and assign the classes to the appropriate cluster set for which the similarity value is higher. Thus, by repeating this process, each class is assigned to a particular cluster.

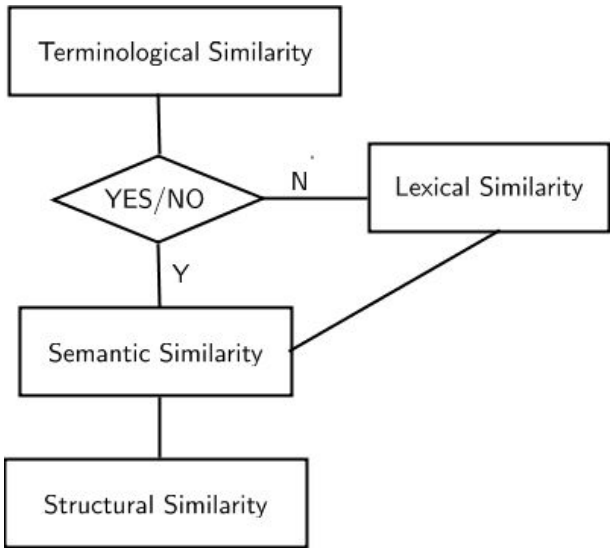


Figure 4.5: Steps of Similarity Matching.

### 4.1.3 Similarity Matching

Integration of ontology is not as simple as just mapping classes in one ontology to corresponding classes in another ontology. Finding matching by maintaining the actual meanings of classes is a great challenge for ontology matcher. So, by keeping this in mind, we have organized our matching task by computing structural similarity and terminological similarity with semantic verification. Fig. 4.5 represents steps of similarity calculation methods used in the proposed method. Description of the matching task is presented below.

#### Terminological Similarity Calculation

The terminological similarity in the proposed method is introduced in order to satisfy the following requirements.

- Strings with small differences should be recognized as being similar. For example the word “Numberofpages” and “Numpage” should remain same.
- Recognize similarity between semantically similar but structurally different strings.

The terminological similarity is calculated in two steps. In the first step, it is checked that whether two strings are identical or not. Here, the term *identical string* means that two strings having the same semantic significance. Again, for dissimilar strings a limit parameter is selected. If the difference between the length of two strings is less than the limit value, then we

select those strings for the next step of terminological similarity calculation. The next step is calculating similarity using WordNet and similarity matching algorithm. Below are the description:

### String Similarity Using Wordnet

The next step is to compute string similarity using WordNet, a lexical database [28]. A word is represented here with its synonyms and meaning of different synonyms. To determine terminological similarity between two classes we take all the synonyms of a class from WordNet and then check availability of other class in the synonym set of first class. Assume two classes, $S_1$ and $S_2$. These two terms are equivalent if $S_2$ contains in the synonym set of $S_1$. The following equation is used to determine terminological similarity.

$$\mathrm{Sim}(S_1, S_2) = \begin{cases} 1 & \text{if } S_2 \text{ is in Synonym}(S_1) \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

For example, in WordNet synonyms of the word paper are report, composition, theme. If ontology $O_1$ contains report and $O_2$ contains paper, We select these types of string as similar string and collect them for next steps. If this step fails to find out similarity between two concepts then the next concept is to find string similarity using edit distance algorithm.

### String Similarity Using Edit Distance Algorithm

In some cases, strings with small differences should be recognized as being similar because they are semantically similar but structurally different. For example, the two words “France” and “French” are semantically similar but structurally different. To handle this type of strings, we find out, how many adjacent character pairs are contained in both the strings. Consider this example,

FRANCE : {FR, RA, AN, NC, CE}

FRENCH : {FR, RE, EN, NC, CH}

Then we find out which character pairs are in the both strings. In this case, the intersection is {FR, NC}. Then the similarity between two strings $S_1$ and $S_2$ is twice the number of character pairs that are common to both the strings divided by the sum of the number of character pairs in the two strings, which is achieved by the following equation.

$$\text{Sim}(S_1, S_2) = \frac{2 \times |\text{pairs}(S_1) \cap \text{pairs}(S_2)|}{|\text{pairs}(S_1)| + |\text{pairs}(S_2)|} \tag{4.5}$$

This formula rates completely dissimilar strings with a similarity value of 0. The other values of the metric always lie between 0 and 1. For our comparison of "FRANCE" and "FRENCH", we found that the value is 0.4. Here, we take a threshold value 0.4, which is rather empirical. We have experimented with different types of strings and found that in most of the cases two strings are similar where comparison value lies between 0.4 to 0.5. In this case, we take the lower boundary of the matching range as threshold value i.e 0.4, so the similarity value above or equal this threshold value is treated as similar string.

**Lexical Similarity Calculation**

If two string are not terminologically similar, as computed above, then we compare again the labels of the two classes. In some cases, the two classes contain same label but different class id's. Since, lexical feature space is the human-readable information stored in label or comment of an entity, if two classes contain same label information then we can easily find out similarity between these two classes. To handle these types of cases, if the label contains a set of strings then all the labels are tokenized. Uppercase and non-alphabet characters indicate where the text is to be split. Then, we find out the common token between the set of split token of two classes. The common token is subdivided with total number of split token set. The following equation is used in this case,

$$\text{Sim}(S_1, S_2) = \frac{\text{token}(l_1) \cap \text{token}(l_2)}{\text{token}(l_1) \cup \text{token}(l_2)} \tag{4.6}$$

The above equation provides the degree of existence of common terms between the labels of two classes. Here, token $l_1$ and token $l_2$ are the tokens split from label of two classes. A set of tokens from label $l$, is obtained by dividing a string at punctuation and separation marks, blank spaces, and uppercase changes. If two labels are identical then the similarity value is 1.0.

Otherwise, the value is between 0.0 to less than 1.0. Here, we take a threshold value of 0.5. So far as our experience gained from performing different kinds of experiment goes, we come to know that, two strings are almost identical if the similarity value is greater than 0.5. Thus, strings with similarity value above this threshold value, are treated as almost similar strings.

### Semantic Similarity Calculation

Continuing from the previous two steps, if two entities are terminologically or lexically similar, or both, then it is not mandatory that each entity represents the same meaning in the ontologies. To ensure that each terminologically similar entity contains approximately the same meaning, we find out semantic similarity or proximity between them. Using the semantic verification [1] strategy, adapted for our purpose as detailed below, we found that, it improves the matching results with the inclusion of the correct matches.

As we calculate similarity for both class and property, so we have introduced different kinds of measurements to find out semantic proximity between classes and properties separately. To verify semantic similarity between classes we have introduced the following measurements by adapting [1].

**Property Similarity Calculation:** Ontology may contain classes having property declaration or without property declaration. The verification measurement discussed here is for the classes having property declaration.

In this measurement, two terminologically similar classes are semantically similar, if they share same type of properties. As we have already stressed, property plays a vital role for classes in ontology. It is our observation that, if two classes share almost the same number of similar properties then they would be semantically equivalent.

For example, `City` is a class of ontology $O_1$ having properties like `name`, `area`, `population`, `sub-region` and `Town` is a class of ontology $O_2$ having properties like`name`, `place`, `population`, `area`, `sub-region`. These two classes contain four similar properties i.e. most of the properties are similar, so we can say that these two classes are semantically similar.

**Common Neighborhood Contradiction:** This verification measurement is for the classes without property declaration. The basic requirement to calculate semantic verification among

this type of classes is, they must be terminologically similar. To find out this type of semantic proximity among terminologically similar entities, we consecutively collect the parent nodes and child nodes of these classes.

Suppose `Essay` is a class of $O_1$ having parent class `Book`, where as `Book` is the subclass of `Product` and `Biography`, `Composition` is the child class of `Essay` . Again `Thesis` is a class of $O_2$ having child class `Article` and subclass of `Volume`. `Volume` is the subclass of `Item`. `Essay` and `Article` are terminologically equivalent and they might be semantically equivalent if there exists any alignment between their parent node or child node. From the above example we found alignment between `Book` and `Volume`,`Composition` and `Article`. Therefore, we found alignment between the parent and child node of two comparable classes, so we can say that these two classes are semantically equivalent.

To verify semantic similarity between properties we have introduced the following measurement.

**Domain and Range Value Incompleteness:** In our proposed method, two properties would be similar if the domain classes of these properties are equal. For example, `author` is a property of ontology $O_1$ and `writer` is a property of ontology $O_2$. Here, `author` and `writer` are terminologically similar but not semantically equivalent. Because, the domain class of first property is `Book` and second class is `Magazine`, which are not terminologically equivalent. In spite of this calculation, we calculate semantic similarity using range value of those properties. Range, as a built in property, links a property to either a class description or a data range. In this case, two properties are semantically equivalent if the range values are equal. For example, `husband` is a property of ontology $O_1$ having domain class `Woman` and range value `Human`. Again, `spouse` is a property of ontology $O_2$ having domain class `Woman` and range value `Human`. Here, we can say, `husband` and `spouse` are semantically equivalent because they share same domain and range values.

### Structural Similarity Calculation

If two entities are terminologically similar, then, we collect all the neighboring entities of the aligned concepts. After this, we check the terminological or lexical similarity between the neighboring concepts as discussed in the above subsection and find out the structurally aligned

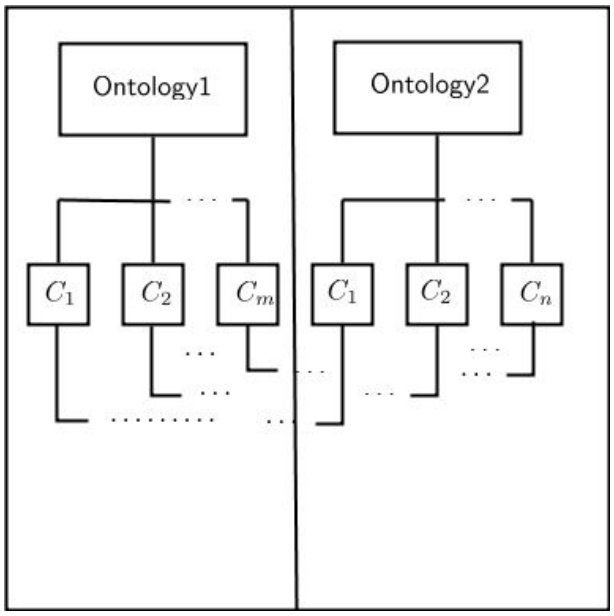


Figure 4.6: Parallel matching task.

concepts. We repeat this process until each neighbor is compared.

### 4.1.4 Parallelization and Distribution of Matching Task

The similarity matching process described above is handled by parallel and distributed matching technique [29]. The main motivation behind this task is that, ontology matching requires a robust and scalable solution that ensures the maximal efficiency of matching operations. This is the main reason for which ontology is partitioned into small clusters. For each of the clusters, similarity matching task is performed in parallel and distributed over the computing resources. Fig. 4.6 describes how to divide the ontologies and match the concepts of each part in parallel with the concepts of a selected ontology.

To execute matching task simultaneously we used parallelism technique in our own way. In the previous section, we have already described the clustering approach, so for similarity matching we started our work with the set of clusters. At the beginning of the parallel algorithm we found the available number of processors to execute services by creating fixed threads. Then, we calculated the number of clusters which participated in each thread to perform the task of similarity matching from total cluster length and available processor. Each thread performed its task simultaneously at the same time. At last, each of the matching tasks is combined to get total number of alignments between classes of ontologies. The process of entity alignment combination is described in the following section.

### 4.1.5 Entity Alignment Determination

Entity alignment combination is an important issue in ontology matching. In our proposed method each of the similarity matching method described above sequentially produces a set of aligned concepts between ontologies. Here, we consider the semantic alignment as the basic alignment process. First, we calculate terminological or lexical similarity of classes and then semantic similarity. If terminologically similar concepts are not semantically aligned, then they are considered as misaligned concepts. If terminologically aligned concepts are semantically similar, then we collect the aligned entity pair into an output alignment set. Then the next step is to calculate the structural similarity of these terminologically similar classes. However, in structural similarity we try to find similarity among neighboring concepts. If a neighboring concept is found as aligned pair among the neighboring concepts of others, then, it is treated as aligned. Otherwise, it is ignored as misaligned concept. Thus these structurally aligned entity is aggregated and inserted into the output alignment set to produce final alignment.

# Chapter 5

# Experiments

We have implemented the proposed approach in Apache Jena [30], a free and open source Java framework for building semantic web and linked data applications. We have applied our proposed method on varieties of datasets, including small ontologies such as Bibliography, Book, Camera, Russia12, TourismAB and large ontologies such as Anatomy, Food, FMA, OpenGALEN and FullGALEN ontology. For better comparison, our experimental study has been performed on data sets, considered in [12] and [15]. All the experiments are carried out on an Intel Duo centrino 1.6 GHz desktop machine with 3 GB DDR2 memory under Windows XP Professional operating system (SP2) and Java 1.7 compiler. In this section, we will discuss the result attained in our experiments on some small and large scale datasets in terms of scalability, memory consumption and time efficiency.

## 5.1 Experiment with Small Ontology

This subsection contains detail description of evaluation datasets with experimental methodology and performance analysis. The performance of the proposed method is measured based on partitioning ontologies as well as on finding alignments between entities.

### 5.1.1 Evaluation Dataset

In our experiment, as mentioned above, we have used Russia12 and TourismAB ontology which have been mentioned in [12]. The reasons for selecting them as the test cases are: 1. they come

from real world domains and widely used in the field of ontology matching, 2. their sizes are moderate. We have also used some other small sized ontologies such as bibliography, book and camera ontology. Short description of the ontologies are given below:

Russia12. The two ontologies cited in [12] are created independently by different people from the content of two travel web sites about Russia. Among them, Russia1 contains 151 classes and 76 properties, Russia2 contains 162 classes and 81 properties. The reference alignment file contains 85 alignments.

TourismAB. The two ontologies cited in [12] are created separately by different communities describing the tourism domain of Mecklenburg–Vorpommern. TourismA contains 340 classes and 97 properties, TourismB contains 474 classes and 100 properties. The reference alignment file contains 226 alignments.

Bibliography[1]. The bibliography ontology contains bibliographic things on the semantic web in rdf. Among them bibliography1 ontology contains 119 classes and 105 properties, bibo1 ontology contains 24 classes and 44 properties.

### 5.1.2 Experimental Methodology

We have conducted three types of experiments in order to ascertain, 1. quality of clustering, 2. correctness of alignment, and 3. compute the scalability, memory consumption, time complexity of finding alignment between ontologies.

The first experiment is done to determine the clustering quality. Because, good clustering quality reflects the effectiveness of the clustering algorithm and good clustering quality ensures the possibility of discovering correct alignments. To determine clustering quality, we manually partitioned the ontology into clusters using close human intervention. After that, we again partitioned them with our proposed clustering approach. Then we compared the resultant clusters with the manual clustering result and determined the correctness of clustering.

Our second experiment is to determine the mapping quality (correctness) of the proposed method. For this reason, we evaluated our method with a popular ontology matcher, FALKON [12]. Because that matcher worked with large ontologies and used clustering method to determine

[1] http://bibliontology.com/

Table 5.1: Table showing clusters found from various small ontologies.

| Ontology | Class Number | Proposed Method | Manual |
|---|---|---|---|
| Russia1 | 151 | 6 | 6 |
| Russia2 | 162 | 6 | 5 |
| TourismA | 340 | 8 | 8 |
| TourismB | 474 | 8 | 8 |

alignment between entities. Moreover, in our third experiment, we experimented to observe the time efficiency, effects on memory consumption and to evaluate scalability of our proposed method.

### 5.1.3 Experimental Result

Experimental results of our proposed method as elaborated below, can be judged based on the aspects which have been mentioned previously.

#### Clustering Quality

According to our first experiment of determining clustering quality the result produced by close human intervention rather than applying proposed clustering method, we found 6 clusters for Russia1; 5 for Russia2; 8 for TourismA and 8 for TourismB. On the other hand, our proposed clustering algorithm has produced almost the same number of clusters for Russia12 ontology. But Russia2 has produced few more clusters than the number of clusters produced by human intervention.

Table 5.1 shows the number of clusters that have been produced from different ontologies. In our method the clusters are not partitioned into smallest blocks like FALKON [12], because we build clusters automatically from ontologies by maintaining internal structural proximities and neighbouring relationship of classes. We have used property-class relationship of classes for clustering purpose which is a new concept. Again, we kept the number of clusters smaller to reduce the computation cost of similarity mapping. Thus, our proposed approach is quite dif-

ferent from the previously mentioned method. Table 5.2 shows comparison result of produced cluster number between FALKON and our proposed method.

### Time Efficiency

If we consider the runtime spent by our method and FALKON during partitioning of ontologies and matching entities, we find that our method is more efficient than FALKON. Based on the current implementation, our method takes around 2 seconds to complete the comparison between Russia12 and 5 seconds for TourismAB, while FALKON spends nearly 5 seconds and 8 seconds correspondingly. Again, we also performed other different experiments over ontologies for testing performance, which are shown in Table 5.3. It may be argued that the machines used and hence the processing powers are different for the aforementioned methods. Still, as we have used parallelism and distribution technique for alignment measurement so our algorithm should take less time to produce alignment between entities.

### Correctness of Alignments

The average number of comparison to find alignment between classes of our proposed method is smaller. Because, once we found alignment between two classes, then according to the nature of the cluster, alignment easily found between their neighbouring classes by collecting and comparing them with each other. Thus the average number of comparisons per aligned pair of our algorithm is approximately determined by total number of class/average number of children. The main motivation of our proposed method is to calculate one-to-many alignment

Table 5.2: Table showing comparison of number of clusters used.

| **Ontology** | **FALKON** | **Proposed Methodology** |
|---|---|---|
| Russia1 | 22 | 6 |
| Russia2 | 22 | 6 |
| TourismA | 18 | 8 |
| TourismB | 23 | 8 |

Table 5.3: Time Efficiency (S) Comparison.

| **Ontology** | **FALKON** | **Proposed Methodology.** |
|---|---|---|
| Russia12 | 5 | 2 |
| TourismAB | 8 | 5 |

between classes. As we compared the correctness of our aligned concepts with FALKON, the comparison results showed that, our method produces better result than FALKON. Our proposed method produced 82 alignment for Russia12 and 167 for TourismAB. If we compare the number of alignments with the total number of reference alignment, then we will find that the correctness of proposed method is 0.91 for Russia12 and 0.91 for TourismAB, while FALKON constructs 0.85 for Russia12 and 0.91 ones for TourismAB. So, the potential computational cost of our proposed method on discovering alignments is smaller than that of FALKON and the number of block mappings is higher than FALKON. The comparison results of alignment correctness are exhibited in Fig. 5.1.

### Memory Efficiency

We have experimented with both large and small sized ontologies. Table 5.4 contains the name of ontologies, memory consumption to store ontology model in memory. We have stored classes, properties and relationship of ontologies in a managed and structured way to minimize the memory consumption. We experimented for retrieving alignments across several pairs of ontologies and we hardly found out of memory exception. As we have used parallelization tech-

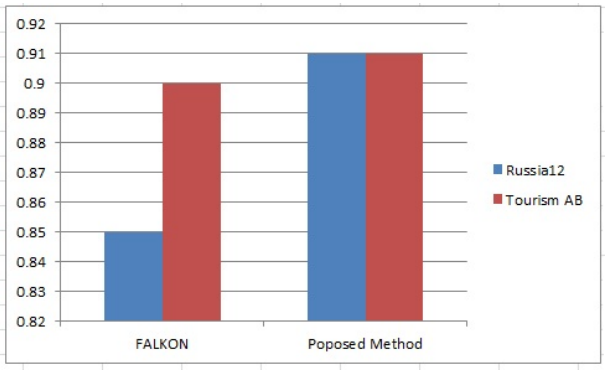


Figure 5.1: Correctness of Alignments comparison.

Table 5.4: Memory Consumption Comparisons.

| **Ontology1** | **Ontology2** | **Memory(MB)** |
|---|---|---|
| Russia1 | Russia2 | 7 |
| TourismA | TourismB | 10 |
| Bibliography | Bibo1 | 2 |

nique so each of the process is distributed within computational resources and increase memory efficiency.

## 5.2 Experiment with Large Ontology

We also experimented with large ontologies for discovering alignments. We have used large ontologies such as Foundational Model of Anatomy (FMA), SNOMED CT, National Cancer Institute Thesaurus (NCI), Full-Galen and OpenGalen for this purpose. The following subsection contains detail description of evaluation datasets with experimental methodology and performance analysis .

### 5.2.1 Evaluation Dataset

In OAEI 2013, the organizer provide large dataset for large ontology matching task. These ontologies are semantically rich and contain ten thousands of classes. We have used these dataset for our experimental purpose. Short description of these ontologies are given below:

FMA[2]. The whole ontology contains 78,989 concepts and 108 properties. It is an evolving computer-based knowledge source for biomedical informatics.

Full-Galen[3]. The whole ontology contains 23,141 concepts and 950 properties. It is an evolving computer-based knowledge source for biomedical informatics.

[2]http://sig.biostr.washington.edu/projects/fm/
[3]http://bioportal.bioontology.org/ontologies/GALEN

SNOMED CT[4]. The ontology contains most comprehensive, multilingual clinical healthcare terminology in the world. It's big overlapping module with FMA and NCI contains 122,464 classes and 110 properties and small overlapping module with NCI contains 51,128 concepts and 102 properties.

NCI[5]. The NCI ontology covers vocabulary for clinical care, translational and basic research, public information and administrative activities. It contains 66,724 classes and 273 properties.

### 5.2.2 Experimental Methodology

For small ontologies, clustering quality is determined by partitioning the ontology into clusters using close human intervention and we found that, our proposed method produced same number of clusters like clusters generated by human intervention. So, we manually partitioned few of the large ontologies into clusters using close human intervention. Then we again partitioned them with our proposed clustering approach. We compared the resultant clusters with the manual clustering result and determined the correctness of clustering. But like small sized ontology we can not compare our method with FALKON for large ontologies. As FALKON used two large ontologies Human and Mouse but they have no property declaration, so we can not properly compare the number of alignments between our method and FALKON in this case.

However, for large ontologies we compare the mapping quality (correctness) of the proposed method with a popular ontology matcher, Anchor Flood [15]. Because that matcher worked with large ontologies and started their work by selecting an anchor from the set of classes. The term anchor means, terminologically similar class and their main alignment algorithm starts from this anchor by exploring its' neighbouring classes. We also conducted our experiment to observe the time efficiency, effects on memory consumption and to evaluate scalability of our proposed method for large ontologies.

[4] http://www.ihtsdo.org/index.php?id=545
[5] http://ncit.nci.nih.gov/

Table 5.5: Table showing clusters found from various large ontologies.

| Ontology | Class Number | Proposed Method | Manual |
|---|---|---|---|
| FMA-nci | 3696 | 42 | 39 |
| NCI-fma | 6488 | 74 | 68 |

### 5.2.3 Experimental Results

Experimental results, as elaborated below, of our proposed method can be judged based on the aspects which have been mentioned previously.

#### Clustering Quality

According to our first experiment of determining clustering quality, we experimented with FMA, SNOMED and NCI ontology for which we used two relatively small fragments of FMA and NCI where the FMA fragment contains 3,696 classes (5% of FMA) and the NCI fragment contains 6,488 classes (10% of NCI), small fragments of FMA and SNOMED wherehe FMA fragment contains 10,157 classes (13% of FMA) and the SNOMED fragment contains 13,412 classes (5% of SNOMED) and small fragments of SNOMED and NCI where the SNOMED fragment contains 51,128 classes (17% of SNOMED), while the NCI fragment contains 23,958 classes (36% of NCI). Table 5.5 represents clusters comparison found from various large ontologies.

Again, in our proposed method we do not start the aligning process by collecting neighbouring concepts from anchor like Anchor-Flood [15]. We build clusters automatically from ontologies by maintaining internal structural proximities and neighbouring relationship between classes. On the other hand, Anchor-Flood started blind search to find out anchor. Their algorithm started if any anchor found. But our method is not dependent on finding out anchor, it divided the whole ontology into smaller cluster to find out the terminologically similarity as well as structurally similarity between classes. Thus, our proposed approach is quite different from [15].

Table 5.6: Number of Aligned Pair Comparison.

| **Ontology1** | **Ontology2** | **Anchor Flood** | **Proposed Methodology** |
|---|---|---|---|
| FMA | Full-Galen | 3056 | 4276 |

Table 5.7: Number of Aligned Pair for Large Ontologies.

| **Ontology1** | **Ontology2** | **Aligned Pair** | **Reference Aligned Pair** |
|---|---|---|---|
| FMA-nci | NCI-fma | 1,450 | 2,890 |
| NCI-snomed | SNOMED-nci | 16,323 | 17,929 |
| FMA-snomed | SNOMED-fma | 7,451 | 8,271 |

**Correctness of Alignments**

The comparison results of number of aligned pair are illustrated in Table 5.6. From the experiment, we found that, our method can produce large number of alignment than alignment produced by above mentioned, Anchor Flood algorithm. The only difference between our proposed method and Anchor Flood method is that, they used subclass of relationship for finding out alignments between ontologies. But our proposed method produced clustering as well as alignments based on property relationship. But despite of these our proposed method produces quite better result than the above mentioned methodology.

We also experimented with other large ontologies and the alignment result is described in Table 5.7. From the experimental result, we found that our proposed method produced almost same number of alignments as the reference alignment mentioned in OAEI 2013.

We again conducted our experiment with the ontology againts itself to observe the nature of the operation over highly similar ontologies. The alignment result and memory consumption are described in Table 5.8 and from the experimental result we found that our proposed method produced maximum number of alignments for highly similar ontologies.

Table 5.8: Number of Aligned Pair for Similar Ontologies.

| Ontology | Number of Aligned Pair | Memory Consumption (MB) |
|---|---|---|
| FMA-nci | 3,600 | 5 |
| NCI-snomed | 15,323 | 12 |
| FMA-snomed | 6,271 | 7 |

Table 5.9: Time Efficiency.

| Ontology1 | Ontology2 | Elapsed Time(S) |
|---|---|---|
| FMA-nci | NCI-fma | 17 |
| NCI-snomed | SNOMED-nci | 90 |
| FMA-snomed | SNOMED-fma | 24 |

**Time Efficiency**

We also performed other different experiments over large ontologies for testing performance, which are figured in Table 5.9. The minimum number of elapsed time, displayed in these tables, depicted the efficiency of our algorithm. As we maintained neighbouring relationships between clusters, so if an alignment found inside a cluster then each of the alignment related to the entity definitely found inside the cluster. As a result, total number of comparison is minimized and our algorithm clearly showed the performance enhancement over other systems.

In Anchor-Flood method they conducted experiments of FMA.owl ontology against itself, which contains around 72 thousand concepts to observe the nature of the operation over highly similar ontologies. To compare our time efficiency in this field we compare the result with Anchor-Flood which is shown in Table 5.10. In Anchor flood, FMA vs. OpenGalen ontology took much time and more average number of comparisons. It is due to the large number of children available in several different nodes. But our proposed method took less time to find out alignment between them. So, our algorithm clearly showed, the performance enhancement over the other systems.

Table 5.10: Time Efficiency (min)Comparison.

| **Ontology1** | **Ontology2** | **Anchor Flood** | **Proposed Methodology** |
|---|---|---|---|
| FMA | FMA | 7.58 | 6.23 |
| FMA | Full-Galen | 3.43 | 2.45 |

Table 5.11: Memory Consumption Comparisons of Large Ontologies.

| **Ontology1** | **Ontology2** | **Memory(MB)** |
|---|---|---|
| FMA - nci | NCI- fma | 12 |
| NCI- snomed | SNOMED - nci | 95 |
| FMA - snomed | SNOMED - fma | 195 |

**Memory Efficiency**

We have experimented with both large and small sized ontologies. Table 5.11 contains the name of ontologies, memory consumption to store ontology model in memory. We have stored classes, properties and relationship of ontologies in a managed and structured way to minimize the memory consumption. We experimented for retrieving alignments across several pairs of ontologies and we hardly found out of memory exception. As we have used parallelization technique so each of the process is distributed within computational resources and increase memory efficiency.

The data contained in Tables 5.10 and 5.11 shows the fact of the scalability of our proposed method. Our algorithm kept the memory consumption low, kept the number of average comparisons stable, and kept the elapsed time linear. Our algorithm, build clusters based on neighbouring relationship. Moreover, our algorithm used parallelism technique within the segment to find alignments and computed similarity values among the concepts of clusters. Thus, the algorithm reduced an average number of comparisons as it restricted within segments. So, our algorithm kept the elapsed time shorter, even when the ontologies are very large.

# Chapter 6

# Conclusion and Future Work

## 6.1 Compendium of Attainments

In our research, we have proposed a new method on ontology matching that can effectively align ontologies of arbitrary size, that helps our method to achieve high performance and scalability over previous alignment algorithms. A brief description of the proposed method is given below.

- Our proposed method can take both large and small ontologies as input. It automatically performed clustering on large ontologies and ignores clustering for smaller ones.

- For each ontology we created our own persistent storage with classes, individuals, properties and relations. From this storage we used subclassof relationship and property-to-class relationship to perform clustering between entities. The basic idea behind clustering is, to perform parallelization and distribution of matching task. After clustering, we performed alignment operation in parallel for each of the entities of clusters, which reduced run time and out of memory exception.

- In similarity matching part, we used terminological, lexical and structural similarity matching method with semantic verification. For terminological and lexical similarity we used, WordNet and string matching algorithm in our own way. For lexical similarity we calculated similarity between the label and comments of the classes in the clusters. We used neighbouring relationship of classes to calculate structural similarity between classes. At final stage, we calculated semantic similarity between classes those having terminological

and structural similarity.

- We experimented our proposed method with two popular ontology matchers and found that our proposed method outperformed in some aspects to other alignment systems.

## 6.2 Future Work

In future, we will work on ontology matching that can find out similarity between ontologies of different languages. Our future target includes, increasing the alignment strength by finding $n : m$ alignment between entities of ontology and implementing a tool for web interface that can find alignment between entities dynamically to work with semantic web services. Finally, we would like to go beyond matching large ontologies and cope with large database schemas and XML schemas in order to support data integration between different data models.

# Bibliography


[1] Y. R. Jean-Mary, E. P. Shironoshita, and M. R. Kabuka, "Ontology matching with semantic verification," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 3, pp. 235–251, 2009.

[2] L. Yu, *A developer's guide to the semantic Web*. Springer, 2011.

[3] "Semantic web." `http://www.w3.org/standards/semanticweb/`.

[4] "Semantic web." `http://http://www.semanticweb.org/wiki/Semantic_Web/`.

[5] L. Yu, *A Developer's Guide to the Semantic Web*. Springer, 2011.

[6] `http://www.ontologymatching.org/`.

[7] J. Euzenat and P. Shvaiko, *Ontology Matching*. Springer, 2013.

[8] P. Shvaiko and J. Euzenat, "Ontology matching: state of the art and future challenges," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, no. 1, pp. 158–176, 2013.

[9] G. Stoilos, G. Stamou, and S. Kollias, "A string metric for ontology alignment," in *The Semantic Web–ISWC 2005*, pp. 624–637, Springer, 2005.

[10] P. Mork and P. A. Bernstein, "Adapting a generic match algorithm to align ontologies of human anatomy," in *Data Engineering, 2004. Proceedings. 20th International Conference on*, pp. 787–790, IEEE, 2004.

[11] P. Lambrix and H. Tan, "Sambo—a system for aligning and merging biomedical ontologies," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 4, no. 3, pp. 196–206, 2006.

[12] W. Hu, Y. Qu, and G. Cheng, "Matching large ontologies: A divide-and-conquer approach," *Data & Knowledge Engineering*, vol. 67, no. 1, pp. 140–160, 2008.

[13] M. Nagy and M. Vargas-Vera, "Towards an automatic semantic data integration: Multi-agent framework approach," *Chapter in Sematic Web. In-Tech Education and Publishing KG*, 2010.

[14] J. Li, J. Tang, Y. Li, and Q. Luo, "Rimom: A dynamic multistrategy ontology alignment framework," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 8, pp. 1218–1232, 2009.

[15] M. H. Seddiqui and M. Aono, "An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 4, pp. 344–356, 2009.

[16] S. Zhang and O. Bodenreider, "Hybrid alignment strategy for anatomical ontologies: Results of the 2007 ontology alignment contest.," in *OM*, 2007.

[17] T. Kirsten, A. Thor, and E. Rahm, "Instance-based matching of large life science ontologies," in *Data Integration in the Life Sciences*, pp. 172–187, Springer, 2007.

[18] A. Gross, M. Hartung, T. Kirsten, and E. Rahm, "On matching large life science ontologies in parallel," in *Data Integration in the Life Sciences*, pp. 35–49, Springer, 2010.

[19] A. Tenschert, M. Assel, A. Cheptsov, and G. Gallizo, "Parallelization and distribution techniques for ontology matching in urban computing environments.," in *Proceedings of the Fourth International Workshop on Ontology Matching*, 2009.

[20] E. Rahm, "Towards large-scale schema and ontology matching," in *Schema matching and mapping*, pp. 3–27, Springer, 2011.

[21] T. Mittra and M. M. Ali, "Ontology matching by applying parallelization and distribution of matching task within clustering environment," in *Electrical and Computer Engineering (ICECE), 2014 International Conference on*, pp. 445–448, IEEE, 2014.

[22] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[23] `http://oaei.ontologymatching.org/2007/`.

[24] A. Valarakos, V. Spiliopoulos, K. Kotis, and G. Vouros, "Automs-f: A java framework for synthesizing ontology mapping methods," in *International Conference i-Know, Graz, Austria*, 2007.

[25] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy, "Representing and reasoning about mappings between domain models," in *AAAI-02 Proceedings*, pp. 80–86, 2002.

[26] M. Erdélyi and J. Abonyi, "Node similarity-based graph clustering and visualization," in *7th International Symposium of Hungarian Researchers on Computational Intelligence*, 2006.

[27] P. Jaccard, "Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines," *Bulletin de la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 241–272, 1901. (in French).

[28] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[29] R. H. Carver and K.-C. Tai, *Modern multithreading: implementing, testing, and debugging multithreaded Java and C++/Pthreads/Win32 programs*. John Wiley & Sons, 2005.

[30] "Apache Jena." `https://jena.apache.org/`.