NeCO: Ontology Alignment using Near-miss Clone Detection

by

Paul Louis Geesaman

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen’s University
Kingston, Ontario, Canada
January 2014

Copyright © Paul Louis Geesaman, 2014
Abstract

The Semantic Web is an endeavour to enhance the web with the ability to represent knowledge. The knowledge is expressed through what are called ontologies. In order to make ontologies useful, it is important to be able to match the knowledge represented in different ontologies. This task is commonly known as ontology alignment. Ontology alignment has been studied, but it remains an open problem with an annual competition dedicated to measure alignment tools’ performance. Many alignment tools are computationally heavy, require training, or are useful in a specific field of study. We propose an ontology alignment method, NeCO, that builds on clone detection techniques to align ontologies. NeCO inherits the clone detection features, and it is light-weight, does not require training, and is useful for any ontology.
Acknowledgments

This thesis would not have been possible without the help and support of those around me. There are too many people to list, as everyone who encouraged me to pursue a Master’s degree helped. However, there are people who I feel it is important to mention their contributions below.

Firstmost, I would like to thank my supervisors, Dr. Jim Cordy and Dr. Amal Zouaq, for their support and guidance. Jim has been very supportive in helping me with NiCad and the low-level technical details. Amal has been a great help to understand ontologies and provided excellent recommendations on how to move forward at all stages of my Masters. The support of my two supervisors made this research possible.

I would like to thank my family for having supported me until I could support myself, helping me get through the difficult parts of a Master’s program, and their support in continuing my education.

I would also like to thank my labmates who provided edits, help with figures, as well as the constructive discussions, and for the occasional distractions. Karolina provided lots of help editing my thesis, and was always keeping me in line. Doug was great for providing help with some technical problems. Eric helped me a lot with administrative matters. Scott provided the kind of insight that can only be gained
through many years of experience. I’d also like to thank Tawhid, Mark, Elizabeth, Amal, Gehan, Charu, Matthew, Andrew for the discussion we had in and out of the lab.

Finally, I would like to thank the professors who hired me as a research assistant while I was an undergraduate student, Dr. Doug Mewhort and Dr. Bennet Murdock. They provided me with the experience that made me fall in love with research.
Contents

Abstract i

Acknowledgments ii

Contents iv

List of Tables vii

List of Figures viii

Chapter 1: Introduction 1
  1.1 Motivation ............................................. 1
  1.2 Objective ............................................ 2
  1.3 Contributions ...................................... 2
  1.4 Outline of the Thesis ............................... 3
  1.5 Summary ............................................ 4

Chapter 2: Background and Related Work 5
  2.1 Ontologies ........................................... 5
  2.2 The Ontology Alignment Problem ................... 8
  2.3 Ontology Alignment Tools ........................... 10
     2.3.1 Background Knowledge .......................... 10
     2.3.2 Rule-based Systems ............................. 12
     2.3.3 Information Retrieval .......................... 13
     2.3.4 String-based Methods .......................... 14
     2.3.5 Graph-based Methods .......................... 14
  2.4 Clone Detection .................................... 15
     2.4.1 Clone Types .................................... 17
     2.4.2 Contextualization .............................. 17
  2.5 Summary ............................................ 18

Chapter 3: Overview 19
List of Tables

9.1 Alignment tool results on the Biblio dataset ........................................ 62
9.2 Traditional clone-detection results ....................................................... 63
9.3 NeCO’s results ................................................................................... 63
9.4 Average for test levels ......................................................................... 65
9.5 Tools’ precision on the 2011 Biblio dataset ............................................ 67
9.6 Tools’ recall for the 2011 dataset ........................................................... 67
9.7 Tools’ F-measure results for the 2011 dataset ........................................ 68
9.8 NeCO’s statistics for different alterations to the ontology ......................... 69
9.9 NeCO results for the 2012 Biblio dataset ................................................ 71
9.10 Statistics for NeCO’s results for the 2012 Biblio Dataset by test level ....... 71
9.11 Tools’ execution time for the 2011 Biblio dataset in minutes .................. 73
## List of Figures

2.1 Example excerpt from an OWL ontology in RDF/XML . . . . . . . 9

3.1 Outline of our alignment method . . . . . . . . . . . . . . . . . . . . 21
3.2 An example OWL ontology . . . . . . . . . . . . . . . . . . . . . . . . 23
3.3 OWL elements extracted from the sample ontology . . . . . . . . . . 24
3.4 Example of the contextualization of the “MastersThesis” OWL class . 26
3.5 Example of clone detection results . . . . . . . . . . . . . . . . . . . . 27
3.6 Best-matches selected from Figure 3.5 . . . . . . . . . . . . . . . . . . 28

4.1 Extraction of OWL classes . . . . . . . . . . . . . . . . . . . . . . . . 30
4.2 Excerpt from the grammar - headers and common elements . . . . 32
4.3 Grammar rules for the extraction of classes . . . . . . . . . . . . . . . 33
4.4 TXL Tranformation main function . . . . . . . . . . . . . . . . . . . . 35
4.5 Extracted classes from Figure 2.1 . . . . . . . . . . . . . . . . . . . . 35

5.1 Contextualization process . . . . . . . . . . . . . . . . . . . . . . . . 37
5.2 OWL classes extracted from the botany ontology . . . . . . . . . . 39
5.3 TXL rules for inlining . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
5.4 Example OWL concept contextualization . . . . . . . . . . . . . . . . 41
5.5 The rule addSourceTags . . . . . . . . . . . . . . . . . . . . . . . . . . 43
Chapter 1

Introduction

In this chapter, we introduce the thesis topic by providing our motivations, contributions, and an overview of the remainder of the thesis.

1.1 Motivation

The Semantic Web is an idea that proposes extending the current day web to present data that can be comprehended by both humans and machines [6]. In order to accomplish this task, the World Wide Web Consortium makes proposals for web standards such as the Extensible Markup Language (XML) [10], the Resource Description Language Schema (RDFS) [11], and the Web Ontology Language (OWL) [4].

Within the Semantic Web context, ontologies are a way to describe knowledge so that the information within an ontology may be cross-referenced with other ontologies to enable machines to reason over this knowledge. Thus, ontologies represent the backbone of the Semantic Web. OWL is the language recommended by the W3C to describe knowledge. It is built on top of XML and RDF [4].

When two ontologies are compared, the task of finding which parts of each ontology correspond to parts of the other is the task of ontology alignment. The task of ontology
1.2 Objective

We approach the alignment problem with a minimalistic approach, in which the alignment is done on large ontologies by using few computational resources. Our approach is to perform fast, computationally-light methods of aligning ontologies, with no prior training, and in a way that can be used on any OWL ontology.

In this thesis, we propose to apply near-miss clone-detection techniques to the task of ontology alignment. We name our tool NeCO, short for Near-miss Clones Ontology Alignment. We present the results obtained by NeCO and compare our results with those of other tools on the same dataset. We then apply NeCO to a new dataset for comparison with our results from the first dataset.

1.3 Contributions

This thesis provides two contributions. First, it proposes that the research done in the field of clone-detection may be beneficial to the ontology alignment problem. We
apply contextualization, a technique from the clone detection community, to a new problem.

Our second contribution is the notion of finding a single best answer from a number of clone pairs by comparing similarity values. This is important for ontology alignment, as the traditional clone-detection approach of returning all pairs over a given threshold is not desirable; rather our task requires us to return the fewest number of possible alignments in order to improve the overall precision.

1.4 Outline of the Thesis

We begin by discussing the background knowledge and a general overview of the way our tool performs alignments. Chapter 2 discusses the background knowledge relevant to the Semantic Web, how other tools align ontologies, and a background on clone detection. This is followed in Chapter 3 by a high-level overview of our alignment method.

Chapters 4 - 7 go into great detail about each step of our alignment method. Chapter 4 describes how our method extracts relevant elements from ontologies for comparison. Chapter 5 describes the task of contextualization, where we replace references to elements with a description of the element. Chapter 6 describes the process of detecting how two extracted fragments of code are flagged as possible alignments. Chapter 7 describes the process of taking all the results from the previous chapter, and finding the best results from the elements returned by traditional clone-detection techniques, providing an alignment.

We discuss the evaluation of our implementation in Chapters 8 and 9. Chapter 8 describes how we evaluated NeCO, and provides a description of the dataset.
Chapter 9 lists the results obtained from NeCO given the experimental conditions described in the previous chapter.

We conclude the thesis with Chapter 10, which presents an overview of what was done, and provides some ideas for future work.

1.5 Summary

This chapter gives an outline for the rest of the thesis and an explanation of why the research done in this thesis is of importance. The next chapter presents the relevant background knowledge, other tools that have been used for ontology alignment, and the topic of clone detection.
Chapter 2

Background and Related Work

In the previous chapter, we presented a short introduction to the thesis, along with an outline to the rest of the thesis. This chapter presents the background material relevant to ontology alignment, clone detection, and some of the methods that have been used to approach this problem and similar problems.

2.1 Ontologies

Data on the web is made so that a machine may present information. The web can link webpages, but it does not have a built-in mechanism to link relevant information contained within the webpages from one source to another. This means that the task of referencing multiple webpages to find the information is done by humans rather than by using a machine to integrate knowledge bases and query multiple websites.

For example, a banking website’s layout may be represented in html, but the legal details are presented in a body of text between html tags. A user can search specific terms to find more information, or read the documentation to understand the terms and conditions of the website. A user may also use an algorithm to search the text for a term and its relevant information, but the website is not made to represent
knowledge; The information requires an agent to interpret the semantics.

Tim Berners-Lee envisions a future where the web becomes more machine-readable, so that a machine may link data across multiple data sources and find answers to queries [6]. He calls the future version of the internet the Semantic Web - where knowledge is presented in a “semantic” manner that is both machine and human-readable for information exchange.

The Semantic Web’s building blocks for representing knowledge are ontologies, which are a collection of information represented in a way to foster machine comprehension. An ontology describes a domain, or a particular subject, such as law, biomedical research or mathematics. The Web Ontology Language (OWL) [4] is the representation of ontologies recommended by World Wide Web Consortium (W3C) and allows for the linking of knowledge from other ontologies written in OWL [63].

OWL is built from DAML+OIL and an extension of the Resource Description Framework (RDF) [4]. It also comes in three varieties: OWL-Lite, a version of OWL with the smallest vocabulary so as to make a simpler reasoning system; OWL-DL, a version of OWL that provides the maximum expressiveness whilst still guaranteeing a computable answer to all queries; and OWL-Full, a version of OWL that uses all aspects of RDF to be understood as an OWL document, however with no guarantee that a given query will be computable [63]. OWL 2, an extension to OWL, is being developed by the W3C for information interchange with the Semantic Web [23, 25]. We do not discuss OWL 2 for this project as results that are significant in OWL should also be significant in OWL 2.

We are primarily interested in doing research with OWL ontologies because OWL has greater expressiveness and reasoning capabilities than RDFS. The syntax of OWL
is often expressed in RDF/XML as shown in Figure 2.1. The components that make up OWL ontologies are: classes, individuals, properties, data values. Classes represent concepts; individuals are instances of a class; properties describe qualities that individuals have; data values can be numbers, strings, or data in some other form.

Properties define restrictions, and values that may be present within a domain. A property may be unrestricted, indicating merely the presence of a property; or carry some restriction, indicating a limit on the values allowable. The properties are either datatype properties, which define a data value for individuals of a class, or object properties, which form relationships between classes so that individuals from their respective classes are linked [4]. Using the example in Figure 2.1, there are two properties: an ObjectProperty named hasSeeds and a DatatypeProperty named Color.

Individuals are instances of an object that are relevant in a domain. Individuals represent a member of a class [63] and may be classified as being part of a single class, or multiple classes. Each individual has the properties and characteristics defined by the class definition under which the individual is classified [39]. For example, “RomaTomatoes” in Figure 2.1 refers to an individual that belongs to the Class “Tomato”.

Classes describe concepts within a domain [4]. A class can be thought of as a set. The set contains individuals described by a domain with the properties and restrictions of the class declared [63]. For example, an ontology for the domain of botany has a class to describe a “Fruit” concept. A class describing the concept of “Tomato” would include an XML subClassOf tag to indicate that every individual that fits under the class “Tomato” is also an individual in the class “Fruit”. The
2.2. THE ONTOLOGY ALIGNMENT PROBLEM

named classes “Tomato” and “Fruit” are shown in Figure 2.1.

For example, “Literature” may be thought of as an example for a domain. “Book”, and “Research Paper” are concepts within the “Literature” domain, thus the concepts “Book” and “Research Paper” are represented in the ontology as classes. An object property for the class “Book” is the existence of an author. A datatype property of the class “Book” is the “Genre” of a book. An individual is an instance of a class. “2001: A Space Odyssey” is an individual that fits under the concept “Book”. The individual can also be assigned a “Genre” value of “Science Fiction”. The individual “2001: A Space Odyssey” would be linked to the individual “Arthur C. Clarke” of the class “Author”.

2.2 The Ontology Alignment Problem

The Semantic Web aims to be an open project where anyone may make changes and additions to ontologies [6]. There may exist many ontologies describing a particular domain or overlapping domains. Differences arise between the ontologies’ descriptions due to the number of people who each describe a domain with differing details, levels of precision, mismatched terminology, and created for dissimilar goals. For example, two lawyers describing “Maritime Law” could make slightly different ontologies despite the fact they are describing the same domain. One lawyer may write an ontology to be far more precise in one aspect of maritime law, another lawyer can write an ontology that describes the domain in a general way, and a third ontology may omit sections of maritime law because other sections are not required for the author’s needs.

Ontology alignment is the process of finding correspondences between elements. Each alignment describes the entities of an ontology that are aligned; the similarity
2.2. THE ONTOLOGY ALIGNMENT PROBLEM

Figure 2.1: Example excerpt from an OWL ontology in RDF/XML
of the elements; and the alignment type: superset, subset, equality or disjoint. For the sake of simplicity, entities are considered disjoint if no alignment is provided. Sometimes, entities are somewhat overlapping but still different. For example, a book editor may have many duties similar to that of a film producer. In aligning the ontologies of a book publisher ontology, and a film studio ontology, the book editor and film producer may be partially aligned. Ontology alignment is an open problem inhibiting the wide-spread adoption of the Semantic Web.

2.3 Ontology Alignment Tools

The Ontology Alignment Evaluation Initiative (OAEI) has made competitions on a yearly basis since 2003 to evaluate ontology alignment tools. The results for the 2011 OAEI competition [20] were available when we began our experiments, which led to the 2011 dataset being chosen for our experimentation. The OAEI has since published the 2012 competition results [1]. In this section, we describe some of the tools used in previous competitions and an overview of how these tools aligned ontologies.

2.3.1 Background Knowledge

One method for ontology matching tools to perform alignments is to use background knowledge, such as a machine-readable lexicon. This technique is quite common, and many of the ontology matchers use information external to the ontology to find semantically related words[13, 16, 27, 42, 43, 58]. Such tools are also known as external techniques, as opposed to internal techniques, which use information contained only within an ontology [27]. Background knowledge techniques are typically used in conjunction with other techniques to obtain alignments.
2.3. ONTOLOGY ALIGNMENT TOOLS

There are a few ways to access background knowledge. WordNet [40] is one such tool for finding synonyms, antonyms, and other lexical information of words. Linked data [7, 60] is an initiative to be able to represent datasets in RDF and OWL and be able to make connections or links between different RDF and OWL datasets. The data should be available on the web, be machine and human-readable, and be written according to W3C standards [5]. Consequently, the combined datasets may be combined with other datasets for querying and drawing inferences. DBpedia [3] is one such source of background knowledge.

DBpedia is an open dataset of information extracted from the infoboxes and links of Wikipedia articles. DBpedia’s information may be combined with other open datasets to create a larger linked network of data.

The “Friend of a Friend” ontology [12] is a linked data set that describes the relationships between agents: people, social groups, organizations, or other groups of people. The idea is to link the agents listed above to each other. An entry of an Agent may include information like their work, membership within groups, interests, and other information about the agent.

WikiMatch is an example ontology alignment tool which uses information from Wikipedia [27]. The tool uses the links between Wikipedia articles of one language to its equivalent in another language to identify its translation in different languages, and Wikipedia’s search function to find similarities between classes, and properties. If the classes or properties across ontologies are above a certain threshold, then the classes or properties are aligned. The matcher compares the terms both Wikipedia articles share in common, and if enough of the terms are the same, they are aligned.
MaasMatch [53] is a matching algorithm that finds the synonyms of every extracted element’s labels. The tool uses WordNet to calculate a similarity based on the synonyms in common between entities, and returns an alignment based on the highest similarities.

### 2.3.2 Rule-based Systems

Rule-based alignment tools use rules to infer ontology alignments [16, 17, 24, 36, 42]. Two examples of rule-based tools are provided below.

CIDER [24] is an ontology matching system that finds lexical similarities in order to be able to infer more about the semantics of the ontology. A similarity value is computed based on the Levenhstein distance of the lexical similarity between labels, and a value based on vector space modelling [46] for the other inputs: similarity of comments, and other lines which make a description; similarity between each term’s hyponyms; similarity between each term’s hypernyms; and the similarity of the properties between words described in the extracted features of an ontology. The alignment tool then proceeds to feed the above mentioned features into the input layer of a neural network. There is a separate neural network for aligning classes, and another for aligning properties. The output layer of the neural network produces a value matrix of similarity values between terms. The tool then finds the highest similarity value for each term. If the highest value is above a specific threshold, then CIDER returns an alignment between the two terms.

AROMA [17] is an algorithm which finds relevant terms to words and database schema matching methods for the use of ontology alignment. The first step of the algorithm is to apply association rules [22]. The tool builds a list of relevant ‘terms’
associated with a ‘concept’, in this case, a class or property. The tool computes a value of how related two concepts are based on their common terms. The tool also creates a hierarchy of terms so that it may relate a concept’s terms to its children and parents to help with alignment. The second step of the algorithm is relating how similar a particular concept and its children are by comparing another concept and its children, and finding the terms in common and taking into account the percentage of terms that both concepts do not share. Also, if two concepts’ parents are aligned with one another, then the children concepts are more likely to be aligned by the tool.

2.3.3 Information Retrieval

Information retrieval techniques find similarity values of related terms even if they have not co-occurred together. Some techniques put terms together into categories [8] which may reflect that the terms have a common grouping. Some alignment tools use information retrieval methods to help provide alignments [28, 53, 58, 59].

One tool that finds results with information retrieval is First, AgreementMaker [16]. AgreementMaker uses a lexicon to compute and extract features of the ontologies, such as longest common substring, to compute a similarity matrix from term frequency - inverse document frequency vectors and their cosine values, as well as other features. The second “layer” matches structural similarities between ontologies. The third step weighs the similarity values from the first and second steps to provide a 1-1, 1-n, or n-m alignment.
2.3. ONTOLOGY ALIGNMENT TOOLS

2.3.4 String-based Methods

There are methods that treat the ontology as text [16, 24, 35] rather than trying to treat OWL files semantically.

Hertuda [26] is a simple matcher that tokenizes strings, then uses a Desmerau-Levenshtein distance to return similarities between entities in the ontologies. If the similarities are above a threshold, then the two entities are aligned.

DDSim [42] creates a similarity measure between every entity of the same type. The similarity is computed with a measure called the Jaccard index, finds hypernyms, and uses mathematical techniques to find values for missing data, and inconsistent data.

2.3.5 Graph-based Methods

Many ontology alignment tools treat the ontology alignment task as a graph-based problem [29, 33, 62]. These methods typically find similarities between some parts of the ontologies, then look at the structure of the ontology as a whole, or at the neighbours of aligned entities.

Ontology Mapping by Evolutionary Programming (MapEVO) and Ontology Mapping using Discrete Particle Swarm Optimisation (MapPSO) [9], treat alignments as optimization problems. MapPSO uses swarm algorithms to find an optimal alignment for individual alignments. MapEVO selects alignments based on a measure of fitness.

MapSSS [13] is a tool that works in two steps, and a last step that had yet to be integrated in the 2011 competition. If a match is completed at any of these steps, the tool performs an alignment on subgraphs resulting from the removal of the aligned element. The first step is a syntactical comparison, which returns exact matches for
elements after some preprocessing. The second step, structural, assumes the previous alignments are correct, then finds if there are any elements which logically must be aligned. For example, if an element has two neighbours, and both neighbours are aligned to one another in both ontologies, then the elements must be aligned.

Optima [58] treats the ontology as a directed graph with named classes for nodes, and edges as properties between classes. For each potential alignment, a single value is derived from syntactical features, using the Smith-Waterman algorithm [56], which obtains a value based on the longest common substrings, single value decomposition to find relationships between terms within WordNet, and other features. The tool generates a matrix where each column represents the classes from one ontology, and the rows represent the classes from the other ontology. A 0 is assigned to each entry in the matrix, representing no alignment. An alignment, changing the value to 1 in the alignment matrix, is determined with Dempster’s expectation-maximization algorithm [18] for the most likely alignments using the information already contained in the alignment matrix using prior probabilities, as well as aligning neighbours of already aligned elements. The expectation maximization algorithm is run iteratively until the tool terminates with a final alignment.

2.4 Clone Detection

A clone is one type of code smell that indicates that code was copied from one source to another. The task of finding fragments of code that were copied from one source to another is the task of clone detection [50].

There are multiple ways to detect software clones [52]. Text-based clone-detection methods compare two code fragments to each other as strings and finds the string
distance between the code fragments [50]. Token-based clone detection uses tokens within the programming language to find clones by comparing two tokens from potential clones. Abstract syntax tree (AST) techniques create a parse-tree for a program, and finds subtrees that correspond to each other. Program Dependency Graph (PDG) techniques take a program’s dependency graph and finds subgraphs that correspond to each other.

We examine the possibility of treating OWL as source-code, and we want to see how effective clone-detection techniques are at aligning ontologies. Clone-detection’s goals are to detect clones that are exact duplicates; have some aspects renamed, but structurally identical; or copied fragments of code with some lines of code added or removed [50].

The detection of clones was used to find Web Service Description Language (WSDL) clones with a process called contextualization [38]. WSDL source code refers to services in other parts of the source code, and in order to reduce the number of false positives, the research contextualized the clones so that the WSDL lines of code that refer to other sections of the source code are replaced by the block of code being referenced. The contextualized fragments of code have more information than non-contextualized fragments for the detection of clones. Metric-based techniques use information about the source code to find clones. Hybrid techniques combine some of the techniques above, and create a new method of finding clones [50].

Delta-p [57] is a tool for comparing software models that transforms UML diagrams into RDF for comparison and for querying.
2.4. CLONE DETECTION

2.4.1 Clone Types

These include a few different types of code clones [50]. There are exact clones, which are code fragments copied from one source to another; renamed clones, which have changes made to the whitespace and identifiers; parameterized clones which have identifiers systematically renamed; near-miss clones with newlines, whitespace, renaming of variables (parameterized and renamed clones are near-miss clones); and gapped clones which have some code removed or inserted. The clones listed above are the clones that are of interest to this project.

Near-miss clone-detection is the process of finding approximate clones of source code. NiCad [51], the near-miss string-based clone detector we use for our results, uses an entire line of code. If a single character between the lines of code is different, then the entire line is flagged as being different. The tool then calculates what percentage of lines are identical. The tool returns a list of code fragments whose percentage of lines are above the threshold, which can be set by the user.

2.4.2 Contextualization

OWL refers to other fragments of code within an OWL file, and as a result, the information is not localized. Because much of the information about a fragment of OWL code is found elsewhere in the OWL file, it is necessary to find the references to other parts of the OWL file, and insert the information so that it may be available to the clone detector.

Contextualization is the process of describing the super-classes of each class in order to describe a class in context. Contextualization was first used for (WSDL) [38], which has the same problems in regards to its non-localized nature. Many classes are
only a single line long, and do not contain enough information to be able to match a class with another class.

2.5 Summary

We reviewed the ontology alignment challenge and the tools that approach the challenge. The next chapter provides an overview of our alignment process, including a high-level description of how our alignment tool works.
Chapter 3

Overview

Chapter 2 detailed some background on the subject, other tools that are used for ontology alignment, and a description of existing clone-detection techniques. In this chapter, we provide an overview of the approach we use for the challenge of ontology alignment.

3.1 The Ontology Alignment Challenge

As previously described, one challenge facing wide-scale use of ontologies is the difficulty in identifying what parts of different ontologies correspond to one another. Ontology alignment, also known as the ontology matching problem, is “... the process of finding relationships or correspondences between elements of different ontologies.” [21]. The task of finding these equivalent parts in ontologies is called the ontology alignment challenge - the problem which we face in this thesis.

For example, a medical doctor may have to describe concepts in medicine, and a biochemist may use an ontology describing biochemistry. These domains have overlapping concepts, and an alignment is desired. The alignment permits the medical doctor to better understand the biochemistry terms and computer programs that
use the medical ontology can use the alignment and biochemistry ontology to their purposes. The ontological components may be named differently, or the ontologies may have differing levels of detail to suit the needs of their respective fields.

As previously described in Chapter 2, a number of solutions have been proposed for the ontology alignment task. The problems with many existing alignment tools are that some tools are slow, some algorithms use large external datasets, use matrix operations which require large amounts of computer resources, or require training, a task that may not always be possible due to the unavailability of training data.

In this thesis we propose a lightweight approach, applying code similarity technology (“clone detection”) to find similar parts of the ontologies as candidates to be aligned. Our process consists of four main phases, as shown in Figure 3.1.

In the first step, the classes of the two ontologies to be aligned are extracted. The second step replaces references to other classes of the ontology by their actual definition to create self-contained fragments - a process called contextualization. The third step puts both contextualized extracted fragments into a clone detector, and finds similar ontology fragments. Step four filters entities in the first and second ontology to find best-matches, returning the final output: a proposed ontology alignment.

Our tool is designed to work on any pair of ontologies in OWL, regardless of the subject of the ontologies, and to return an answer in a minimal amount of time. Each step of our ontology alignment method is described in this chapter. Subsequent chapters go into detail of each step, represented by ovals in Figure 3.1.
3.1. THE ONTOLOGY ALIGNMENT CHALLENGE

Figure 3.1: Outline of our alignment method
3.2 Extraction of OWL elements

Clone detection requires a target granularity to be extracted and compared for similarity. For example, code blocks, functions or classes are extracted when using programming languages and represent different granularities. In the ontology alignment problem, the units to be aligned are classes, properties and individuals. Each extracted element is called a potential clone. This thesis presents the findings on the extraction and alignment of classes. The first step of our process is the identification and extraction of the OWL class elements as potential clones.

Figure 3.2 shows a simple example OWL ontology, and Figure 3.3 the OWL classes extracted from this sample ontology. In the example from Figure 3.3, each potential clone is contained within a source tag. Each clone tag keeps track of the potential clone’s original file, and the location in the file with the startline and endline attributes.

The extracted elements are passed into the next step of the algorithm.

3.3 Contextualization

Similarity using clone detection techniques can only work when all of the relevant attributes of the units to be compared are directly present in the unit representation. Unfortunately, most of the attributes of OWL elements are described in OWL notation by reference, using references to their attributes defined elsewhere in the ontology. One challenge we face is to increase the information in each class for the task of ontology alignment.

As highlighted earlier, contextualization [38] is a process previously described for the WSDL modelling language, in which referenced attributes are copied to their
Figure 3.2: An example OWL ontology
Figure 3.3: OWL elements extracted from the sample ontology
references in order to localize attributes and thus make the units to be compared amenable to clone analysis.

The second step in our process contextualizes our extracted OWL elements in this way, recursively inlining referenced OWL classes of each extracted OWL element to localize all of its attributes. Figure 3.4 shows the result of contextualization of the extracted elements of Figure 3.3.

After the contextualization step, each element has all the information contained within it to provide a full description, rather than referring to other constructs.

3.4 Near-Miss Clone-Detection

Once the elements to be compared have been extracted and contextualized, we search them for clones using near-miss clone detection. Near-miss clone detection is a method for analyzing similarity in textual code fragments in which small differences are ignored, allowing for variations in the representation. This corresponds well to the alignment problem - two ontologies for the same concepts may vary in the level of detail and may use different naming conventions, in much the same way that implementations of the same function in different web browsers’ source code may vary in statement ordering, variable naming, comments and so on.

Near-miss clone-detection identifies sets of similar fragments called clone pairs. For any particular code fragment, there may be many other code fragments which are similar to varying degrees, up to the threshold difference. In our first experiments, we treat all of the OWL elements identified as similar to an element as potential alignments. While this can yield a high level of recall, that is to say we find most of the possibilities; it gives us a very low precision, that is, we give too many possible
Figure 3.4: Example of the contextualization of the “MastersThesis” OWL class
answers, which is not as useful. Figure 3.5 shows the result of using clone detection to find all of the elements in our example that are similar to another element.

The `<clone>` tag shows the details of a clone pair. A similarity value is shown for each potential pair of fragments of ontologies within the `<clone>` tag. Within the `<source>` tag, the clone detector assigns an individual identification number for each potential clone, the `pcid` attribute.

The next step of our algorithm approaches how we reduce the number of answers returned so as to have better predictive power.

### 3.5 Best-Match Filtering

As previously explained, using near-miss clone-detection alone for the task of ontology alignment yields too many results, proposing an alignment with very poor precision. Step four of our algorithm addresses this issue by filtering the results of the clone detection to identify only the set of similar elements that have the highest similarity, ignoring those of lesser similarity. This technique, called best-match filtering, yields a
3.6 EXPERIMENT

Figure 3.6: Best-matches selected from Figure 3.5

much more precise set of potential alignments - possibly at the cost of missing other good ones.

Figure 3.6 shows the result of filtering the clone detection results in Figure 3.5 to yield only the best match results, to be specific, the clone pair with the highest similarity.

3.6 Experiment

Using the best-match clone detection technique, we evaluated our implementation of our method, NeCO, against existing ontology alignment tools in Chapter 9.

3.7 Summary

In this chapter we have outlined our new process for ontology alignment based on nearmiss clone detection. Using a running example, we trace the steps of our process and give a quick overview of the steps. In the following chapters, we describe the details of each step and experiment to compare our method to other ontology alignment tools.
Chapter 4

OWL Class Extraction

In the previous chapter, we provided an overview of the method we developed to approach the ontology alignment problem. We discussed the steps in a high-level manner. In this chapter we present TXL grammars to describe OWL documents and the extraction rules for finding the OWL classes of interest. This process implements the extraction shown in Figure 4.1, which shows the input, an ontology combined with an OWL grammar and TXL rules which produce the output, the extracted OWL classes.

4.1 TXL

NiCad [51], a tool we are using to compute a similarity value discussed in Chapter 6, requires certain format of the input. In order to extract named classes, we must choose a language which can extract text with patterns. This step can be done with programming languages such as XSLT or Perl, however we used a programming language called TXL [14]. TXL is a rule-based programming language designed for text transformation. We chose this programming language because our clone detection technique requires that rules be written in TXL to extract the relevant information.
4.1. TXL

The clone detection technique will be discussed in Chapter 6.

4.1.1 Grammar

The use of TXL for the extraction of OWL classes requires a grammar for identifying the OWL fragments of interest. OWL’s syntax was written in as a TXL grammar in order to be able to parse an OWL ontology [4]. The grammar can determine what granularity should be extracted. In this research, we are only concerned with extracting named classes. A class can be represented in one of many different ways [4]. A named class is a class with a URI. Other classes include an enumeration of individuals.
that make up a class, a list of restrictions on properties, the intersection of multiple classes, the union of multiple classes, and the complement of a class. The alignment of the other 5 types of classes are left for future work. The ontology alignment problem refers to the alignment of classes, properties, and individuals, however in this work we only align named classes.

The first type of class can be referenced by its URI with the `subClassOf` tag. A `subClassOf` tag indicates that all the individuals belonging to a particular class also belong to its parent classes [11].

Other granularities can be specified, including parts of classes, multiple classes, instances, and properties. The alignment of properties and instances is beyond the scope of this thesis, however the grammar parses them for future work. Figure 4.2 contains an excerpt of the grammar created for OWL ontologies.

The grammars for TXL are context-free grammars made of terminal and non-terminal rules [14]. Rules are created with the `define` keyword, with non-terminal tokens in `[ ]`. Every grammar requires a base non-terminal `program`, then a set of rules defined for the specific language. In Figure 4.2, the OWL grammar contains the information expressed at the beginning of every OWL file: `namespaces`, `doctypes`, `xmlVersion`. Moreover, defining the pattern for comments was necessary for the removal of all comments.

TXL removes the comments when running the program, a task most clone detection techniques perform [50], including the clone detection technique we use. Figure 4.2 shows a part of the grammar generated for OWL.

The most important part for this experiment is how the grammar identifies named classes, the non-terminal element `namedClass`. Figure 4.3 shows some of the grammar
define program
    [xmlVersion][NL]
    [opt namespaces][ontologyHeaders]
end define

comments
    <!-- -->
    <rdfs:comment> </rdfs:comment>
end comments

define namespaces
    '<!DOCTYPE [opt doctypes] '[ [NL][IN]
    [repeat entity][EX]
    ']> [NL]
end define

define doctypes
    [SP] [doctypesSpecific] [SP]
end define

define xmlVersion
    '<?xml [SP] 'version= [stringlit] [opt encoding] '?>
end define

define entity
    '<!ENTITY [SP] [id] [SP] [stringlit] [SP] '>[NL]
end define

define ontologyHeaders
    < 'rdf:RDF [NL][IN]
    [repeat headers] [SP] '>[NL]
    [repeat axiom] [EX]
    </ 'rdf:RDF> [NL]
end define

Figure 4.2: Excerpt from the grammar - headers and common elements
Figure 4.3: Grammar rules for the extraction of classes
that deals with classes.

A class element can have two forms: `classID` and `classAbout`. The difference between these two forms lies in the identifying attribute, whether the class is named with the `rdf:ID` attribute, or the `rdf:about` attribute. The `rdf:ID` attribute refers to a local designation, and `rdf:about` refers to an absolute URI. For this thesis, we change all `rdf:about` attributes to `rdf:ID` under an assumption that the classes are local, or otherwise are contained within a different ontology.

### 4.1.2 Transformation Rules

The transformation rules format the fragments of OWL so that they may be analyzed by the clone detector, explained in chapter 6. Figure 4.4 shows the main function that is run by TXL. The function calls rules to prepare the data for extraction, extract the classes, and inline the program, discussed in Chapter 5.

Before the extraction of classes, all `rdf:about` attributes are changed to `rdf:ID` attributes, represented in Figure 4.4 as the `[AboutToID]` rule. This is done to avoid the need to create extra rules for extraction, so that all named classes are assumed to have `rdf:ID` attributes.

The line `_[^ NewP]` within Figure 4.4 finds all fragments of code that fit under `classID` and keep them stored in a variable called `ClassesWithoutPreprocessing`. The contents of `ClassesWithoutPreprocessing` are shown in Figure 4.5, which concludes the extraction process.
function main
    replace [program]
        P [program]
    construct NewP [program]
        P [convertSingletons][AboutToID]
    construct ClassesWithoutPreprocessing [repeat classID]
        _ [^ NewP]
    construct ClassesToInline [repeat classID]
        _ [^ NewP]
    construct InlinedProgram [program]
        ClassesToInline [inline ClassesWithoutPreprocessing]
        by
        InlinedProgram [addSourceTags][removeDuplicates]
end function

Figure 4.4: TXL Tranformation main function

<owl:Class rdf:ID="Fruit">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&BotanyExample;hasSeeds"/>
      <owl:someValuesFrom rdf:resource="&BotanyExample;Seeds"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Tomato">
  <rdfs:subClassOf rdf:resource="&BotanyExample;Fruit"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&BotanyExample;Color"/>
      <owl:someValuesFrom rdf:resource="&xsd;string"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

Figure 4.5: Extracted classes from Figure 2.1
4.1.3 Extracted OWL Classes

Figure 4.5 shows the final step of the extraction process. Each `<owl:Class>` tag is a named class from the ontology. In this step, we found the named classes that could be aligned to another class from another ontology.

4.2 Summary

In this chapter, we described the input, the grammar created to describe an OWL ontology, and began explaining the transformation rules used by TXL to extract elements. These steps encompass the procedures shown in Figure 4.1. The next chapter discusses the contextualization process, the manner in which we localize the information contained within a particular extracted fragment of OWL code, and how we formatted the code fragments to be used by a clone detection method, discussed in chapter 6.
Chapter 5

Contextualizing Ontology Entities

In the previous chapter, we discussed OWL ontologies and elements of interest for ontology alignment, the grammar created for TXL to describe OWL, and the extraction process. In this chapter, we explain the process of contextualization. This process means replacing references with their actual contents. The contextualization process adds information contained within the ontology so that there is more information for a given class. This step is shown in Figure 5.1
5.1 Contextualization

Elements in OWL ontologies often refer to other elements within the same or another OWL ontology. For instance, a class that is a subclass of another class may only be a single line long. The referencing poses problems for clone detection (see Chapter 6), because not enough information is available for making alignments. As discussed in Chapter 2, one way to circumvent this problem is to use external sources to gather more information about the referenced elements; however, gathering information from external sources is typically slower than using the information contained within an ontology. The way we approach this problem is through contextualization.

An OWL ontology contains references to other elements within the ontology. When a line referring its parent is encountered, we replaced that particular line with a description of its parent. As a result, OWL classes contain more lines of code and hence more information for clone detection and, in turn, ontology alignment.

Figure 5.2 shows the input for this process of our ontology alignment method.

5.2 TXL Contextualization Rules

The <rdfs:subClassOf> tag can refer to other classes within the ontology. We create TXL rules to find the classes that <rdfs:subClassOf> refers to, and to replace those lines with the OWL code that constructs them.

We developed TXL rules to find all the subClass0f tags with an rdf:resource attribute referring to a named class. The rules then found the entire class being referenced, and replace of the subClass0f tag with the referenced named class.

The rule is shown in Figure 5.3. In TXL, a rule is applied to a text file until its conditions can no longer be met. The replace [subClass0f] keyword defines a
5.3. CONTEXTUALIZED EXAMPLE

Figure 5.2: OWL classes extracted from the botany ontology

pattern that must be changed. The deconstruct ClassResource,
construct SubClassID [stringlit], and construct IDtoMatch [rdfId] sections
find and construct a string for the class being referred. The section
deconstruct * [classID] AllClasses finds referenced class before having done
any contextualization. Finally, by describes the final output of the inlining rule.

5.3 Contextualized example

Figure 5.2 shows extracted fragments. The extracted fragments represent the
classes that can be aligned to another element from another ontology.

Figure 5.4 shows the class “Tomato”. “Tomato” contains two <subClassOf>
tags. The subClassOf tag with attribute rdf:resource="&BotanyExample;Fruit",
rule inline AllClasses [repeat classID]
  % Find each subclass reference
  replace [subClassOf]
  <rdfs: subClassOf ClassResource [rdfResource] />

  % Get the name of the referred class
  deconstruct ClassResource
  rdf: resource= SubClassIDWithHash [stringlit]
  construct SubClassID [stringlit]
  SubClassIDWithHash [deleteHashPrefix]
  construct IDtoMatch [rdfId]
  rdf: ID= SubClassID

  % Find the definition of that class in the set of all classes
  deconstruct * [classID] AllClasses
  BeginFilename [srcfilename] BeginLinenumber [srclinenumber]
  <owl: Class IDtoMatch >
  S [repeat statement]
  EndFilename [srcfilename] EndLinenumber [srclinenumber]
  </owl: Class>

  % Inline the class definition in the subclass reference
  by
  <rdfs: subClassOf>
  <owl: Class rdf: ID= SubClassID >
  S
  </owl: Class>
  </rdfs: subClassOf>
end rule

Figure 5.3: TXL rules for inlining

refers to another named class within the ontology. The <subClass0f> tag without
an attribute contains local property unique to “Tomato”.

The <rdfs:subClassOf rdf:resource="&BotanyExample;Fruit"/> tag refers
to the class “Fruit”. The contextualization changes the single-line subClass0f tag to
a multiple line description of the subclass with the class definition , and puts between
the two tags the lines describing “Fruit”. The second <rdfs:subClass0f> tag does
5.3. CONTEXTUALIZED EXAMPLE

(a) Unmodified OWL class

```
<owl:Class rdf:about="&BotanyExample;Tomato">
  <rdfs:subClassOf rdf:resource="&BotanyExample;Fruit"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&BotanyExample;Color"/>
      <owl:someValuesFrom rdf:resource="&xsd;string"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

(b) A contextualized OWL class

```
<owl:Class rdf:about="&BotanyExample;Tomato">
  <rdfs:subClassOf rdf:resource="&BotanyExample;Fruit">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&BotanyExample;hasSeeds"/>
        <owl:someValuesFrom rdf:resource="&BotanyExample;Seeds"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&BotanyExample;Color"/>
      <owl:someValuesFrom rdf:resource="&xsd;string"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 5.4: Example OWL concept contextualization
not refer to any outside elements, and it is fully contextualized. The class “Fruit” does not contain any `<rdfs:subClassOf>` lines that refer to other classes, and hence it is fully contextualized.

### 5.4 Adding Source Tags

In order to use the output of this section for the clone detector, discussed in Chapter 6, we must prepare the data with source tags. This is done with the TXL rule `addSourceTags` shown in Figure 5.5.

The rule `addSourceTags` is applied once to every extracted class. The rule finds the lines where a named class appears in the source file, and encloses the fragment of code within a `source` tag. The `source` tag has information about the file contained within its attributes. The `file` attribute contains the name of the file containing the code fragment. The `startline` and `endline` attributes contain information about the line number where the code fragment begins and ends.

An example of the output obtained by rules in Figure 5.5 applied on the code from Figure 5.2 can be seen in Figure 5.6.

### 5.5 Summary

In this chapter, we described the process of contextualization. Contextualization is useful for the task of ontology alignment because it is a way of increasing the information available for the clone detector. In Chapter 6, we present clone detectors, and the particular clone detector we use for our experiment.
rule addSourceTags
    % Find each class, exactly once
    replace $ [classID]
    BeginSrcFilename[srcfilename] BeginSrcLinenumbersrclinenumber
    <owl:Class rdf:ID= IDofClass [stringlit] >
    Statements [repeat statement]
    EndSrcFilename[srcfilename] EndSrcLinenumbersrclinenumber
    </owl:Class>

    % Make sure the class is not empty
    deconstruct not BeginSrcFilename
    % empty

    % Get the class’ original source coordinates in the OWL source file
    construct BeginFilenameString [stringlit]
    _ [quote BeginSrcFilename]
    construct BeginSrcLineString [stringlit]
    _ [quote BeginSrcLinenumber]
    construct EndSrcLineString [stringlit]
    _ [quote EndSrcLinenumber]

    % Wrap the class in a <source> tag
    by
    <source file= BeginFilenameString startline= BeginSrcLineString
    endline= EndSrcLineString >
    <owl:Class rdf:ID= IDofClass >
    Statements
    </owl:Class>
    </source>
end rule

Figure 5.5: The rule addSourceTags
Figure 5.6: Extracted classes with source tags
Chapter 6

Detecting Concept Clones

In the previous chapter, we outlined the need for contextualization and the way we contextualize classes. Contextualization alters the extracted fragments of code, by replacing lines that reference another construct with the actual construct, allowing a fragment to have all the information available within the ontology of the extracted classes to be contained within the class. This chapter examines the detection phase of our method. We use NiCad, a near-miss clone-detection tool, on contextualized classes to find similar OWL entities across ontologies which we treat as possible alignments. We explain how NiCad returns a list of potential alignments.

6.1 Clone Detection

Clone detection is the process of finding software clones, a fragment of source code that has been copied [52]. A clone detector, a tool used for the purpose of clone detection, uses an algorithm to find software clones. An extraction is performed to take all fragments of code within a single project, or between projects, which are called potential clones. When two potential clones are identified by a clone detector as being possible software clones, the clone detector returns the match as a clone pair.
Near-miss clones is one particular type of code clone [50]. Near-miss clone-detection has been used on source code projects to find fragments of code that have been copied from one section of a project to another with modifications [51]. For this reason, our idea is to approach ontology alignment as a near-miss clone-detection problem and discover whether treating the document as source-code would be capable of providing good ontology alignments.

There are several ways to approach the detection of software clones, including token-based, tree-based, and metric-based techniques [50]. Each method has its own
benefits, however to approach the problem of ontology alignment we chose a text-based technique because it is light-weight, and text-based techniques perform better than other techniques if elements of the ontology are removed.

### 6.2 Longest Common Substring Problem

The *Longest Common Substring Problem* takes two strings as input, and seeks to find a third string which is the longest possible substring that can be found in both strings. An example is illustrated in Figure 6.2. Both strings contain `a b c d` in that order, in a non-consecutive way.

A brute-force attempt would enumerate all possible substrings to find the longest substring that both strings have in common, however this method has a worst-case complexity of $O(2^n)$, where $n$ is the number of characters in the larger string [15].

However, the Longest Common Substring Problem meets the two requirements for a dynamic programming solution: the problem can be split into sub-problems and the sub-problems are overlapping [15].

**String 1:**
```
a x b c y z d
```

**String 2:**
```
a b w c d
```

**Longest Common Substring:**
```
a b c d
```

*Figure 6.2: Two strings for finding a common substring*
6.3 NiCad

NiCad [51] is a near-miss clone detector that uses text-based techniques to identify clones. The tool is used to find intentional clones, clones that were copied on purpose from one source to another with changes.

NiCad treats the ontology, a graph, as text. As a result, the alignment problem becomes a longest common substring problem instead of a graph isomorphism problem. The advantage of using the longest common substring algorithm is that the algorithm is fast, whereas it is not known whether there exists an algorithm that can solve the graph isomorphism problem in polynomial time.

The traditional longest common substring algorithm has been modified to return the length of the longest common substring rather than the substring. NiCad treats a line in a file as being the atomic unit of comparison, which are defined by the grammar file with the conventions of pretty-printing [30, 51]. In our grammar, each XML tag represents a single line. The NiCad algorithm takes a threshold parameter, which is the maximum percentage of lines that can be different for two potential clones to be considered a clone pair. NiCad saves time by rejecting the pairing of two potential clones if too many of their lines are different.

NiCad uses the extracted elements from the files as potential clones. It calculates the percentage of lines that are identical, and returns the set of pairs of potential clones that have a high enough percentage of identical lines of code. NiCad returns the set as a list of clone pairs with a similarity value for each pair.

We developed scripts to run all the conditions of the dataset so as to perform many experiments. The results are stored in a file to be accessed later.
This chapter described clone detection, as well as the algorithms used for NiCad. NiCad allows us to find elements across ontologies which may correspond to each other, called a clone pair, and assigns a similarity value for each clone pair. In Chapter 7, we discuss how we reduce noise by filtering the results so that we only obtain clone pairs that are of interest to ontology alignment.
Chapter 7

Filtering Process

The previous chapter described the process of clone detection for finding clone pairs, how NiCad works, and the process of finding code clones. This chapter describes how we were able to improve our filtering of the results that NiCad returns so that our process returns fewer clone pairs that are incorrect. The result of filtering the clone detector’s returned clone pairs to obtain the pair with highest similarity is called a best-match alignment.

7.1 Filtering

When finding code clones, it may make sense to return many potential clones, as a developer is not limited to making a single code clone. However, the nature of software clones is different than that of ontology alignment.

A threshold changes the number of potential clones returned by a clone detector. A high threshold is required when there are many changes across the clones, and a low threshold is required if there are few changes. If a threshold is set too high, then too many potential clones are returned, and if the threshold is set too low, too few potential clones are returned. For this reason, an optimal threshold is necessary.
A single alignment is preferable to returning multiple alignments, which indicates that the threshold must be low, however the ontologies being compared may have many changes, which requires a higher threshold. In order to create a general-purpose alignment tool, we must find a way to combine the advantages of aligning elements that are different, but return as few of them as possible. To achieve this goal, we create the filtering process. The filter returns the best clone pair, that is to say the clone pairs that are most similar to each other. We label the result of the filtering process the \textit{best-match} alignment.

Figure 7.1 gives a high-level view of the filtering process. The left side of the figure has the clone pairs returned from Chapter 6. In this figure, there are three clone pairs
7.1. FILTERING

returned. The topmost clone pairs show the black potential clone is matched with the green potential clone with a similarity of 85. The middle clone pair shows the black potential clone is matched with the blue potential clone with a similarity of 77. The bottom clone pair is the purple potential clone matched with the blue potential clone with a similarity of 87. Assuming that the black potential clone should be matched with the green potential clone, and the purple potential clone should be matched with the blue potential clone, then the goal is to filter out the clone pair that matches the black potential clone with the blue potential clone.

Our early results gave us a reason for thinking that using clone detection as it is used on code would not be useful for ontology alignment because using NiCad without any modifications did not yield accurate alignments. Traditional clone-detection would identify a particular entity in the source file as having multiple entities in the target file. Because the entities were being contextualized, the entities would be aligned with the siblings of the gold standard’s proposed alignment. The problem became more pronounced when a class contextualized many classes. If the class being contextualized contains another class that is large, then an incorrect alignment would occur.

In Figure 7.2, both reported potential clones would be identified as being aligned even though the clone with the higher similarity is more likely to be the correct clone. Figure 7.2 shows a clone report returned from NiCad, in which there are two clone pairs, each contained within the clone XML tag. The attributes of the clone tag include two attributes, most importantly the similarity attribute which indicates the percentage of lines of the potential clones that are identical. Within the clone tags, there are two source tags. Each source tag contains information about the
7.1. FILTERING

<clone nlines="132" similarity="100">
  <source file="Onto1.owl" startline="424" endline="459" pcid="19"></source>
  <source file="Onto2.owl" startline="424" endline="459" pcid="52"></source>
</clone>

<clone nlines="132" similarity="78">
  <source file="Onto1.owl" startline="424" endline="459" pcid="19"></source>
  <source file="Onto2.owl" startline="206" endline="222" pcid="38"></source>
</clone>

Figure 7.2: Clone report returned by NiCad

ontologies being aligned, with additional information about the element being aligned. The **pcid** attribute gives a unique identifier to each potential clone. Lowering the threshold could remove alignments that are correct, and raising the threshold NiCad to return too many clone pairs. Our goal was to remove clone pairs that were not of interest from NiCad’s results. The experimental conditions for evaluating our tool are described in Chapter 8 and the results are further detailed in Chapter 9.

The filtering of NiCad’s results returns the best-match clone pairs. For each concept in the source ontology, we return the set of the concepts in the target ontology which has the highest similarity value in the target ontology. If there is more than one possible answer with the highest similarity value, then all the similarity values that match the highest value are returned. Returning multiple alignments is done because there is no way to determine which of the two clone pairs with the highest similarity is the better answer. In adding this extra step, our alignment method removes noise and improves accuracy.

Alignments specify whether an element from the source ontology is is equal, more general, or less general than the element from the target ontology. For the sake of simplicity, each alignment is assumed to be an equal alignment, that is to say the entity from the source ontology is the same as the entity in the target ontology. There is also a confidence level that is between 0 and 1, the confidence level is set to 1 for
all alignments [21]. The use of other types of alignments and confidence levels are something to be addressed in future work.

For example, Figure 7.2 shows two clone pairs that refer to the same potential clone, as is evident by having the same pcid attribute. In each clone pair, the target ontology’s potential clone is different, as each target ontology’s pcid is different. The similarities of the first and second clone pairs are different; the first clone pair’s similarity is higher than the second clone pair’s similarity. The filtering process would find that the highest similarity for pcid="19" is 100. The first clone pair matches the highest similarity value for pcid="19", but the second clone pair has a lower similarity, so it not returned as an alignment.

7.2 Summary

This chapter presented how we filter NiCad’s results. Each clone pair after filtering returns a best-match clone, which is treated as an ontology alignment. The next chapter outlines the details of our experiment. We describe the dataset we used for experimentation, as well as our results for differing thresholds, traditional clone-detection, and best-match clone-detection.
Chapter 8

An Experiment

In the previous chapter, we described the filtering process, the final step of our alignment method. This chapter describes the design of our experiment which consists of a description of the dataset, the measurements we use, the gold standard, and how we evaluate NeCO.

8.1 The Datasets

Datasets for the evaluation of ontology alignment tools are provided by the Ontology Alignment Evaluation Initiative (OAEI)\(^1\). The datasets allow ontology alignment researchers to evaluate the effectiveness, as well as the weaknesses, of their tools. These weaknesses may consist of difficulty in aligning ontologies from a particular domains or with a type of alteration to the dataset. There are websites for each year’s evaluation\(^2,3\) where the dataset is made available, and so that previous competitions may be attempted. The OAEI seeks to make a standard evaluation dataset to improve

\(^1\)http://oaei.ontologymatching.org/
\(^2\)http://oaei.ontologymatching.org/2011/
\(^3\)http://oaei.ontologymatching.org/2012/
ontology alignment techniques and present findings at conferences\(^4\). The existence of a standard dataset allows us to evaluate our tool, and gives us a benchmark through which we can compare our alignment method with other tools.

We took particular interest in the Biblio dataset, one of many datasets through which we can evaluate our alignment tool. The Biblio dataset is generated by an automatic test data generation tool [48]. Each test transforms a source ontology systematically, such as changing all labels to ontology elements to a random string. Using this dataset allows us to evaluate our alignment method’s strengths and weaknesses. Each test is an ontology with one or many transformations performed on the source ontology, and it is assigned a number so that someone testing a tool may use the corresponding reference alignment to verify on which tests the task performs well, and which transformations on the seed ontology make the tool perform poorly. We began by using the 2011 version of the dataset which provides tests for aligning the domain of ‘bibliography’.

### 8.2 Measurements

There are three standard statistics used to evaluate the effectiveness of an ontology alignment method: precision, recall, and F-measure. These metrics are used for the task of ontology alignment [20].

Precision is a percentage of how many correct alignments are returned by the alignment algorithm. As shown in the equation, a low precision means a tool providing an alignment has many returned alignments, but only a low percentage of the alignments are actual alignments. A high precision means that the tool has provided

\(^4\)http://oaei.ontologymatching.org/
8.2. MEASUREMENTS

a high number of correct alignments with a low number of incorrect alignments.

\[
\text{precision} = \frac{|\{\text{Filtered Elements}\} \cap \{\text{Gold Standard's Pairs}\}|}{|\{\text{Filtered Elements}\}|}
\]

Recall is a percentage of how many correct alignments the tool discovered from the gold standard’s alignments. A low recall occurs if a tool discovers few of the alignments provided by the gold standard, whereas a high recall is obtained if a tool discovers many of the alignments.

\[
\text{recall} = \frac{|\{\text{Filtered Elements}\} \cap \{\text{Gold Standard's Pairs}\}|}{|\{\text{Gold Standard Pairs}\}|}
\]

A tool may have high precision and low recall, in which case there are few returned alignments, but a high percentage of the alignments are correct. A low precision and high recall method returns many alignments, but few of the alignments are of interest.

The special case in calculating precision and recall occurs when NeCO does not return any alignments for a particular test. When this happens, the precision for that particular test is considered to be undefined, therefore we do not apply this precision value to the average precision. However, the recall is 0, and these tests are included for the average recall.

\textit{F-measure} is the harmonic mean of precision and recall. If precision or recall are low, then F-measure will be low as well. However, if precision and recall are equal, then F-measure will be equal to both these measures. For this reason, we decided that F-measure is the most important measure of how successful an alignment had been.
8.3. GOLD STANDARD

A gold standard is provided in both html, displayed in Figure 8.2, and RDF formats, shown in Figure 8.1, for the 2011 dataset.

We used the html file for the evaluation of our alignment. The RDF file, shown in Figure 8.1, could also be used for evaluation, but for our script which computed the evaluation, we chose to use the html file. The script opens the html alignment, and creates a hash table where the left-hand side of the equals sign is the key, and the right-hand side of the equals sign is the value. For each alignment, the script finds
8.4 EXPERIMENT CONDITIONS

The goal of this thesis is to assess the feasibility of using clone detection techniques for ontology alignment. With this in mind, we seek to measure how well NeCO can align classes in ontologies, and to compare how well the alignment of classes compares to other tools which are trying to align all elements of OWL ontologies. The alignment of object properties, datatype properties, and individuals are left for future work.

NeCO uses NiCad, which has parameters. The most important parameter is the ability to select a threshold for evaluation. In order to determine the most effective threshold for evaluation, we tried many thresholds.

A parameter we kept consistent among all trials was the minimum number of lines that a potential clone had to have in order to be evaluated by NiCad. We set the minimum number of lines to 3 as there were some cases where the class would be 3 lines long before contextualization, and would be more lines after contextualization.

For a baseline, we ran the dataset with and without the filter. We compare the results from NeCO and the results from contextualized and unfiltered NiCad alignments to compare the two conditions. We also ran the best parameters for the
2011 dataset on the 2012 dataset to verify whether the results are comparable for the two datasets.

8.5 Summary

This chapter discusses the experimental conditions we set for the ontology alignment. We outline how we measure the tool’s results, as well as describe the dataset. The next chapter describes the results of the experiment using the experimental conditions explained in this chapter.
Chapter 9

Results

In the previous chapter, we described the details of our experiment. This chapter presents the results of our experiments on the 2011 Biblio dataset, along with a comparison of NeCO to other algorithms. We then ran NeCO on the 2012 Biblio dataset and report those results.

9.1 2011 Ontology Alignment Results

The Ontology Alignment Evaluation Initiative (OAEI) publishes results for each year they present data. The results of the tools that entered the competition are available for us to use in comparison with our tool [20]. The dataset used for a baseline is the Biblio dataset. This dataset has a base ontology which describes bibliographies. The base ontology has systematic alterations performed on it to create a new, modified ontology describing the same subject matter. The base ontology and its modified version are aligned.

Table 9.1 shows the results for the tools that entered the 2011 competition, and provided us with a benchmark with which to evaluate our tool.

We performed two tests to compare with the tools shown in Table 9.1. We ran
<table>
<thead>
<tr>
<th></th>
<th>edna</th>
<th></th>
<th>AgreementMaker</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>100-level</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>200-level</td>
<td>0.50</td>
<td>0.51</td>
<td>0.51</td>
<td>0.98</td>
</tr>
<tr>
<td>H-mean</td>
<td>0.50</td>
<td>0.51</td>
<td>0.52</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>Aroma</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.50</td>
<td>0.68</td>
<td>0.53</td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td>0.93</td>
<td>0.68</td>
<td>0.53</td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>CIDER</td>
<td>0.96</td>
<td>1.00</td>
<td>0.96</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>0.89</td>
<td>0.70</td>
<td>0.58</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>0.89</td>
<td>0.70</td>
<td>0.58</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>LDOA</td>
<td>0.77</td>
<td>0.86</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>LDOA</td>
<td>0.99</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>0.54</td>
<td>0.31</td>
<td>0.21</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>0.55</td>
<td>0.32</td>
<td>0.22</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>MapEvo</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.99</td>
<td>0.66</td>
<td>0.49</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>0.99</td>
<td>0.67</td>
<td>0.50</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>MapPSO</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.96</td>
<td>0.77</td>
<td>0.64</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>0.96</td>
<td>0.77</td>
<td>0.64</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>Optima</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.59</td>
<td>0.55</td>
<td>0.52</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>0.60</td>
<td>0.56</td>
<td>0.53</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Table 9.1: Alignment tool results on the Biblio dataset
### 9.1. 2011 ONTOLOGY ALIGNMENT RESULTS

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Precision</th>
<th>F-measure</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>1.00</td>
<td>0.06</td>
<td>0.03</td>
</tr>
<tr>
<td>0.10</td>
<td>0.88</td>
<td>0.09</td>
<td>0.05</td>
</tr>
<tr>
<td>0.20</td>
<td>0.74</td>
<td>0.35</td>
<td>0.23</td>
</tr>
<tr>
<td>0.30</td>
<td>0.40</td>
<td>0.43</td>
<td>0.46</td>
</tr>
<tr>
<td>0.40</td>
<td>0.15</td>
<td>0.23</td>
<td>0.51</td>
</tr>
<tr>
<td>0.50</td>
<td>0.07</td>
<td>0.12</td>
<td>0.53</td>
</tr>
<tr>
<td>0.60</td>
<td>0.05</td>
<td>0.09</td>
<td>0.54</td>
</tr>
<tr>
<td>0.70</td>
<td>0.03</td>
<td>0.06</td>
<td>0.55</td>
</tr>
<tr>
<td>0.80</td>
<td>0.05</td>
<td>0.09</td>
<td>0.61</td>
</tr>
<tr>
<td>0.90</td>
<td>0.03</td>
<td>0.06</td>
<td>0.72</td>
</tr>
</tbody>
</table>

Table 9.2: Traditional clone-detection results

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Precision</th>
<th>F-measure</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>1.00</td>
<td>0.06</td>
<td>0.03</td>
</tr>
<tr>
<td>0.10</td>
<td>0.95</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>0.20</td>
<td>0.87</td>
<td>0.36</td>
<td>0.23</td>
</tr>
<tr>
<td><strong>0.30</strong></td>
<td><strong>0.78</strong></td>
<td><strong>0.56</strong></td>
<td><strong>0.44</strong></td>
</tr>
<tr>
<td>0.40</td>
<td>0.69</td>
<td>0.56</td>
<td>0.47</td>
</tr>
<tr>
<td>0.50</td>
<td>0.65</td>
<td>0.55</td>
<td>0.48</td>
</tr>
<tr>
<td>0.60</td>
<td>0.65</td>
<td>0.55</td>
<td>0.48</td>
</tr>
<tr>
<td>0.70</td>
<td>0.65</td>
<td>0.55</td>
<td>0.48</td>
</tr>
<tr>
<td>0.80</td>
<td>0.60</td>
<td>0.56</td>
<td>0.53</td>
</tr>
<tr>
<td>0.90</td>
<td>0.49</td>
<td>0.54</td>
<td>0.61</td>
</tr>
</tbody>
</table>

Table 9.3: NeCO’s results

Traditional clone-detection without contextualization and NeCO with varying thresholds to find the best threshold. We then compare the two methods to verify whether filtering is a practical step, or whether near-miss clone detection without filtering works for ontology alignment.

Traditional Clone detection’s results without filtering are shown in Table 9.2 and Figure 9.1. NeCO’s results for all the tests are shown in Table 9.3 and Figure 9.2.
9.1. 2011 ONTOLOGY ALIGNMENT RESULTS

Figure 9.1: Traditional clone detection’s statistics for varying thresholds

Figure 9.2: NeCO’s statistics for varying thresholds
9.1. 2011 ONTOLOGY ALIGNMENT RESULTS

<table>
<thead>
<tr>
<th>Tests</th>
<th>Clone Detection</th>
<th>NeCO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>F-measure</td>
</tr>
<tr>
<td>100-level</td>
<td>0.19</td>
<td>0.32</td>
</tr>
<tr>
<td>200-level</td>
<td>0.41</td>
<td>0.42</td>
</tr>
<tr>
<td>H-mean</td>
<td>0.40</td>
<td>0.43</td>
</tr>
</tbody>
</table>

Table 9.4: Average for test levels

We consider F-measure to be the best sign of success of a particular threshold. F-measure is a harmonic mean between precision and recall. The best threshold for both traditional clone-detection and NeCO is 0.30 according to our experiments shown in Table 9.2 and Table 9.3. By comparing the results for Table 9.2 and Table 9.3, we find that the filtering process greatly increases precision with a small cost to recall when the threshold is 0.30. Because of the filtering step, NeCO has a higher F-measure, and can therefore be considered the better method of finding clones.

The 100-level and 200-level tests results are shown in Table 9.4 for comparison with other tools. The threshold traditional clone detection and NeCO are set to 0.30 as they provided the highest F-measure for all the tests.

We compare clone-detection and NeCO to the other alignment tools shown in Table 9.1 in a chart shown in Figure 9.3. Each entry on the chart is one tool’s precision and recall for the 2011 dataset. NeCO’s results are similar to many tools, yielding a better precision than some; however NeCO has worse recall than many of the other tools.

Table 9.5 shows the precision of the tools sorted in descending order. Clone detection and NeCO are shown in bold with the threshold value set to 0.30. The best-match alignment tool has a precision that is comparable to other alignment tools. However, the traditional clone detection technique has the worst precision of
all the alignment tools. Table 9.5 shows that the filtering process improves precision. Table 9.6 shows the recall of all the tools entered in the 2011 competition. The recall of our method is quite low, and it shows the greatest weakness of our method. Finally, Table 9.7 shows the F-measure for our tool and the other tools that entered the 2011 OAEI competition. NeCO performs better than some tools, and a lot better than traditional clone detection.
### 9.1. 2011 ONTOLOGY ALIGNMENT RESULTS

#### Tools sorted by Precision

<table>
<thead>
<tr>
<th>Tool</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogMap</td>
<td>0.99</td>
</tr>
<tr>
<td>MassMatch</td>
<td>0.99</td>
</tr>
<tr>
<td>AgreementMaker</td>
<td>0.98</td>
</tr>
<tr>
<td>YAM++</td>
<td>0.97</td>
</tr>
<tr>
<td>Aroma</td>
<td>0.93</td>
</tr>
<tr>
<td>CODI</td>
<td>0.93</td>
</tr>
<tr>
<td>Lily</td>
<td>0.93</td>
</tr>
<tr>
<td>CIDER</td>
<td>0.89</td>
</tr>
<tr>
<td>CSA</td>
<td>0.89</td>
</tr>
<tr>
<td>MapSSS</td>
<td>0.80</td>
</tr>
<tr>
<td><strong>NeCO</strong></td>
<td><strong>0.79</strong></td>
</tr>
<tr>
<td>Optima</td>
<td>0.60</td>
</tr>
<tr>
<td>MapEvo</td>
<td>0.54</td>
</tr>
<tr>
<td>LDOA</td>
<td>0.51</td>
</tr>
<tr>
<td>edna</td>
<td>0.50</td>
</tr>
<tr>
<td>Traditional Clone Detection</td>
<td>0.30</td>
</tr>
</tbody>
</table>

Table 9.5: Tools’ precision on the 2011 Biblio dataset

#### Tools sorted by Recall

<table>
<thead>
<tr>
<th>Tool</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSA</td>
<td>0.65</td>
</tr>
<tr>
<td>MapSSS</td>
<td>0.62</td>
</tr>
<tr>
<td>CODI</td>
<td>0.60</td>
</tr>
<tr>
<td>YAM++</td>
<td>0.60</td>
</tr>
<tr>
<td>CIDER</td>
<td>0.58</td>
</tr>
<tr>
<td>Lily</td>
<td>0.57</td>
</tr>
<tr>
<td>AgreementMaker</td>
<td>0.56</td>
</tr>
<tr>
<td>Aroma</td>
<td>0.53</td>
</tr>
<tr>
<td>Optima</td>
<td>0.53</td>
</tr>
<tr>
<td>edna</td>
<td>0.52</td>
</tr>
<tr>
<td>LDOA</td>
<td>0.51</td>
</tr>
<tr>
<td>LogMap</td>
<td>0.50</td>
</tr>
<tr>
<td><strong>Traditional Clone Detection</strong></td>
<td><strong>0.45</strong></td>
</tr>
<tr>
<td><strong>NeCO</strong></td>
<td><strong>0.44</strong></td>
</tr>
<tr>
<td>MassMatch</td>
<td>0.44</td>
</tr>
<tr>
<td>MapEvo</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Table 9.6: Tools’ recall for the 2011 dataset
9.2. ANALYSIS

The changes made to the source ontology were changes to the labels attached to classes, either by changing the labels to a random string or translating the ontology’s labels into French. Likewise, the removal of comments, any references of properties, and the removal of individuals are possible alterations made to the seed ontology for the generation of tests. Another systematic change done to the ontology was changing the hierarchy of the ontology, either by flattening, expanding, or suppressing the hierarchy. Finally, some tests included the removal of restrictions within classes, which left no lines of code within the extracted classes.

<table>
<thead>
<tr>
<th>Tools sorted by F-measure</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapSSS</td>
<td>0.77</td>
</tr>
<tr>
<td>CODI</td>
<td>0.74</td>
</tr>
<tr>
<td>YAM++</td>
<td>0.74</td>
</tr>
<tr>
<td>CSA</td>
<td>0.73</td>
</tr>
<tr>
<td>AgreementMaker</td>
<td>0.71</td>
</tr>
<tr>
<td>CIDER</td>
<td>0.70</td>
</tr>
<tr>
<td>Lily</td>
<td>0.70</td>
</tr>
<tr>
<td>Aroma</td>
<td>0.68</td>
</tr>
<tr>
<td>LogMap</td>
<td>0.67</td>
</tr>
<tr>
<td>MassMatch</td>
<td>0.61</td>
</tr>
<tr>
<td><strong>NeCO</strong></td>
<td><strong>0.56</strong></td>
</tr>
<tr>
<td>Optima</td>
<td>0.56</td>
</tr>
<tr>
<td>edna</td>
<td>0.51</td>
</tr>
<tr>
<td>LDOA</td>
<td>0.51</td>
</tr>
<tr>
<td><strong>Traditional Clone Detection</strong></td>
<td><strong>0.43</strong></td>
</tr>
<tr>
<td>MapEvo</td>
<td>0.32</td>
</tr>
</tbody>
</table>

Table 9.7: Tools’ F-measure results for the 2011 dataset
9.2. ANALYSIS

<table>
<thead>
<tr>
<th>Modification</th>
<th>Precision</th>
<th>F-measure</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labels Changed</td>
<td>0.78</td>
<td>0.56</td>
<td>0.44</td>
</tr>
<tr>
<td>Comments Removed</td>
<td>0.77</td>
<td>0.52</td>
<td>0.39</td>
</tr>
<tr>
<td>Instances Removed</td>
<td>0.75</td>
<td>0.52</td>
<td>0.40</td>
</tr>
<tr>
<td>Flattened Hierarchy</td>
<td>0.76</td>
<td>0.66</td>
<td>0.58</td>
</tr>
<tr>
<td>Expanded Hierarchy</td>
<td>0.52</td>
<td>0.32</td>
<td>0.23</td>
</tr>
<tr>
<td>Suppressed Hierarchy</td>
<td>0.81</td>
<td>0.16</td>
<td>0.09</td>
</tr>
<tr>
<td>Property Restrictions Removed</td>
<td>Undefined</td>
<td>Undefined</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 9.8: NeCO’s statistics for different alterations to the ontology

There are 102 tests in the 2011 dataset made up of the changes previously mentioned, and different combinations of these changes\(^1\). An overview of the transformations performed on an ontology and the resulting changes in NeCO’s precision and recall using NeCO are shown in Table 9.8.

The removal of comments did not pose any problems for NeCO because it removes any comments automatically. Changing labels of classes and properties led to a decrease in precision, however no decrease in recall was recorded due to that change. NeCO is able to find alignments due to the structural similarities between the original class, and its altered version. The removal of instances also did not cause any problems, because we do not work on aligning instances in this version of the tool.

Changes to the Biblio hierarchy changes the structure of the graph representing the ontology. The ontology is represented as a graph without cycles, and changes to the structure of the structure representing the ontology had varying effects. Flattening the hierarchy is reducing the number of vertices in the graph; expanding the hierarchy is introducing intermediate classes to increase the number of vertices; suppressing the hierarchy is to change the structure of the graph so that every element is the child of

\(^1\)http://oaei.ontologymatching.org/2011/
9.2. ANALYSIS

the OWL construct called “Thing”. All changes in precision for these conditions can be explained by the changing of the structure of extracted elements.

Flattening in the ontology led to an increase in recall. Expanding the hierarchy, introducing intermediary classes, caused a reduction in precision and recall. We believe the precision and recall fall because the increase in the number of intermediary classes lowers the similarity value of the correct alignment, and increases the likelihood that another element in the target ontology may be incorrectly aligned with the element. The addition of lines of OWL code may also decrease the similarity of correct alignments below the threshold, which leads to a decrease in recall. Suppressing the hierarchy, removing all hierarchical information, caused great difficulty to NeCO with a slightly higher precision, and a very low recall. We believe the similarity value of elements falls substantially, which leads to many potential alignments falling below the threshold, which then leads to a fall in recall. Much of NeCO’s information comes from contextualization, so suppressing the hierarchy of the ontology means that NeCO cannot properly contextualize elements.

The suppressing of restrictions on classes was a test that caused a lot of difficulty for our tool. Classes are defined as a set of individuals, and when defining classes, one must include the restrictions of that class. Because the restrictions were removed, there was no information for the clone detector to align ontologies. NeCO did not return any alignments for tests that include the suppressing of restrictions.
9.3 2012 ONTOLOGY ALIGNMENT RESULTS

The 2012 Biblio dataset introduced some changes, and is available from the OAEI 2012 website\(^2\). More tools participated in the 2012 dataset [1]. We concentrate on comparing the 2012 results of our tool with the 2011 results.

We ran our tool with the thresholds 0.30, and 0.40 because 0.30 provided our best results, but 0.40 was not much worse. Our goal was to examine whether the results from the 2011 dataset would be comparable to the results in the 2012 dataset. The 2012 dataset uses the same source ontology with some syntactical changes. The meaning is unchanged.

The 2012 dataset contained a lot more of the tests on which we did poorly, and removed the easier tests from the 2011 dataset. The changes to the ontology are the same we describe in Section 9.2. Our results are shown in Table 9.9 for the thresholds 0.30 and 0.40.

Table 9.10 is a list of how well the tool performs for the 100 and 200 level tests. This is included to compare the tool to other tools that we showed in Table 9.1.

Our results fell considerably for the 2012 dataset. We found that a threshold of

\(^2\)http://oaei.ontologymatching.org/2012/
0.40 was best, but we could not replicate the success we had with the 2011 dataset.

9.4 Run-time

We cannot directly compare the run-times of the OAEI competitions with the run-time of our method as we do not have access to a computer that is of the same specifications, we run different operating systems, and the scope of our alignments is different. However, it makes sense to indicate our execution time given that our initial requirement was to achieve a quick alignment.

The run-time of our method on a machine with a 2.2 GHz Intel Core i7 processor with 4 GB of RAM running MacOS X version 10.7.5 is approximately 50 seconds. The tests done for the 2011 alignment ran on a machine with a 3GHz Xeon 5472 processor with 8 GB of RAM running Linux Fedora 8. Our machine is not as powerful as the machine that ran the evaluation, however we only perform alignments on a small part of the ontology.

Table 9.11 shows the run-time for the Biblio dataset in minutes. Some tools did not parse all the ontologies or failed to complete, and for these reasons, the run-time is not included [20]. Because our tool took less than one minute on the weaker machine, we are confident that the tool would run in a short amount of time, perhaps even less than the fastest tool in the table.

9.5 Blind Renaming

Blind renaming refers to the process of ignoring differences in identifiers when comparing potential clones in clone detection. Typically, blind renaming leads to a significant increase in recall at the cost of precision. In some applications, such as software model
9.6. SIMILARITY RANGE

<table>
<thead>
<tr>
<th>System</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neco</td>
<td>0.83</td>
</tr>
<tr>
<td>edna</td>
<td>1.06</td>
</tr>
<tr>
<td>Aroma</td>
<td>1.10</td>
</tr>
<tr>
<td>LogMap</td>
<td>2.16</td>
</tr>
<tr>
<td>CSA</td>
<td>2.61</td>
</tr>
<tr>
<td>YAM++</td>
<td>6.68</td>
</tr>
<tr>
<td>MapEvo</td>
<td>7.44</td>
</tr>
<tr>
<td>Lily</td>
<td>8.76</td>
</tr>
<tr>
<td>CIDER</td>
<td>30.30</td>
</tr>
<tr>
<td>MassMatch</td>
<td>36.06</td>
</tr>
<tr>
<td>Optima</td>
<td>149</td>
</tr>
<tr>
<td>MapPso</td>
<td>185</td>
</tr>
<tr>
<td>LDOA</td>
<td>1020</td>
</tr>
</tbody>
</table>

Table 9.11: Tools’ execution time for the 2011 Biblio dataset in minutes

clones, blind renaming yields much higher overall accuracy.

As an additional experiment, we attempted to perform an alignment with blind renaming. However, in our case this technique leads to a decrease in both precision and recall. Recall was decreased due to the best-match process, where the correct answer was rejected because it did not have the highest similarity value.

9.6 Similarity Range

We also attempted to make ranges for matches. Rather than picking a best-match, we made a method of finding a range which we called \( \alpha \). At first, we tried an absolute range, so that we would return all answers that have a similarity higher or equal to the difference between the highest similarity and \( \alpha \). For example, if \( \alpha \) was set at 5%, and the highest similarity for a particular element was 90%, then every alignment above 85% was returned. We found that precision decreased, with an increase in recall, however f-measure decreased.
Next, we performed a relative range. Alpha was set to be a percentage of the difference between 100 and the highest similarity. This was called beta. For example, if beta was 10%, and the highest similarity was 80% for a particular element, then alpha would be set to 2%, and an alignment would return all alignments that had a similarity above 78%. Precision decreased to a smaller degree than our previous experiment, and recall increased. F-measure fluctuated with different values for alpha and beta, but remained about the same.

Finally, we created an offset variable. The idea was that if two elements have high similarity, then a specific alignment is more certain. We used the alpha and beta variables from the previous paragraph, but the offset was to say that if two elements had a similarity above the offset, then one shouldn’t use a range. For example, with an offset of 10%, alpha at 10%, and beta at 10%, if the highest alignment is 90%, then the values for alpha and beta were not used. However, for the a particular element from the source ontology had a highest similarity value of 80% with the same parameters as the situation above, then the range would accept all alignments with similarity 79% and above. The results of this experiment were similar to the ones above, where F-measure fluctuated with different values, but did not show a noticeable improvement.

9.7 Summary

This chapter provided the results of our experiment. We compared the results of using clone detection for alignment with and without the filtering process. We found that the filtering process increases F-measure by increasing precision with a small cost to recall. We conclude that filtering results provides for better alignments.
Chapter 10

Summary and Conclusions

The previous chapter presented our method of evaluating our alignment tool, and the settings we used for evaluation. This chapter summarizes the work, recapitulates the results obtained by NeCO, discusses the limitations of our method, and indicates directions for future work.

10.1 Summary

In this thesis, we presented an ontology alignment technique with the use of clone detection techniques. We postulated that clone detection techniques can be used as a general-purpose ontology alignment technique that is fast to use and does not require any prior training.

Our technique consists of 4 phases: extraction, contextualization, clone detection, and filtering. We implement the method, and call it NeCO, the Near-miss Clone Ontology Alignment tool. We measure NeCO’s precision, recall, and F-measure of the results of the 2011 dataset, then run the 2012 dataset with the same parameters for comparison.

During the extraction phase, we prepare the data for evaluation, so that it can be
used with NiCad. The extraction is done with rules written in TXL which finds all the elements of interest within an ontology, and stores the elements for analysis by NiCad.

The next phase is the contextualization phase. The extracted elements are contextualized, meaning the references to other elements are replaced with definitions of the elements being referenced. NiCad is good at making matches with source code projects, and the attempt to find WSDL clones with contextualization had been successful [38], another language which contains references to outside constructs. These contextualized elements are fed into NiCad.

Next, both ontologies’ contextualized classes are fed into a clone-detector. NiCad runs with a threshold and finds all the elements from each ontology with a percentage of different lines below the threshold. This step returns clone pairs, however these results return too many possible alignments.

To improve the accuracy of the alignment tool, results are filtered so that only the best alignments are returned. The best alignments are taken from NiCad’s output, and by finding the highest similarity value for a particular element in the source ontology.

We compared our results of aligning classes to the results of other tools’ alignments of classes, properties, and individuals; we found that our tool was able to perform alignments reasonably well. We found that NeCO has difficulty aligning elements if the hierarchy of an ontology is suppressed or expanded. These changes to the structure removed much of the information that NeCO requires to work properly.

NeCO does well when aligning ontologies that are actually near-miss clones of each other. For example, if there is an original ontology, and two different authors
have supplemented the original ontology, then both modified versions can be aligned
with each other using this method. More work has to be done to determine whether
it can become a general purpose ontology alignment tool for ontologies that are not
near-miss clones of each other.

10.2 Contributions

This thesis provides two contributions. This thesis proposes that the research done
in the field of clone-detection may be beneficial to the ontology alignment problem.
We use contextualization, a technique from the clone detection community to a new
problem.

Our second contribution is the notion of finding a single best answer from a number
of clone pairs by comparing similarity values. This is important for ontology align-
ment, as the traditional clone-detection approach of returning all pairs over a given
threshold is not desirable; rather our task requires us to return the fewest number of
possible alignments in order to improve the overall precision.

10.3 Limitations

The efficiency of our method is limited if a source ontology’s hierarchy is altered.
This is because NeCO uses the structure of extracted elements for alignment. NeCO
uses the information contained within the contextualized elements, however if these
elements are heavily altered, then NeCO does not have a way to flag two elements as
being aligned.
10.4 Future Work

Near-miss clone detection was developed for use of intentional software clones [51]. As shown in Chapter 9, our method works quite well to align ontologies that are modified versions of each other without altering the structure of the ontology. Based on those results we see the following directions for future work:

- Extending the method to align other elements of interest, such as datatype properties, object properties, and individuals.
- Improving the contextualization process by contextualizing every reference to a class, rather than finding the `subClassOf` statements for contextualization.
- Contextualizing datatype and object properties references with the definition of the referenced properties.
- Ordering the XML tags so that tag order does not matter [2].
- Examining the use of consistent renaming.
- Testing ontologies that are modified versions of each other.


[59] Quang-Vinh Tran, Ryutaro Ichise, and Bao-Quoc Ho. Cluster-based similarity 
Workshop, pages 142–147, 2011.


[63] Chris Welty, Deborah L McGuinness, and Michael K Smith. OWL web ontology 
Appendix A

The 2011 Biblio Source Ontology

This is a graphical representation of the seed ontology used in our experiments.¹

¹http://oaei.ontologymatching.org/2011/benchmarks/101/onto.rdf