APPROVAL SHEET

Title of Thesis: Text Based Similarity Metrics and Delta for Semantic Web Graphs

Name of Candidate: Krishnamurthy Koduvayur Viswanathan
MS Computer Science, 2010

Thesis and Abstract Approved: ________________________
Dr. Tim Finin
Professor
Department of Computer Science and Electrical Engineering

Date Approved: ______________________________

Curriculum Vitae

Name: Krishnamurthy Koduvayur Viswanathan

Permanent Address: 4804 Fernley Square, Baltimore, MD - 21227, USA

Degree and date to be conferred: MS Computer Science, August 2010

Date of Birth: May 7, 1984

Place of Birth: Ernakulam, Kerala, INDIA

Secondary Education: SIES College of Arts, Science, and Commerce, Mumbai,

Maharashtra, INDIA

Collegiate institutions attended:

University of Maryland Baltimore County, MS Computer Science, 2010.
Veermata Jijabai Technological Institute, BE, Computer Science and Engineering, 2006

Major: Computer Science

Professional positions held:

Software Development Engineer Intern for Xerox Global Services Docs Platform Team Webster, NY, May 2009 Aug 2009

Software Development Engineer: Citius IT Solutions, Mumbai (www.citiustech.com), October 2007 to Aug 2008.

Software Development Engineer: MAQ Software India, Mumbai, (www.maqsoftware.com) August 2006 to September 2007.

ABSTRACT

Title of Thesis: Text Based Similarity Metrics and Delta for Semantic Web Graphs

Krishnamurthy Koduvayur Viswanathan, MS Computer Science, 2010

Thesis directed by: Dr. Tim Finin
Department of Computer Science and
Electrical Engineering

Recognizing that two semantic web documents or graphs are similar, and characterizing their differences is useful in many tasks, including retrieval, updating, version control and knowledge base editing. We describe a number of text based similarity metrics that characterize the relation between semantic web graphs and evaluate these metrics for three specific cases of similarity that we have identified: similarity in classes and properties used while differing only in literal content, difference only in base-URI, and versioning relationship.

In addition to determining the similarity between two Semantic Web graphs, we generate a 'delta' between graphs that have been identified as having a versioning relationship. The delta consists of triples to be added or removed from one to make them equivalent. This method takes into account the text of the RDF graph's serialization as a document, rather than relying solely on the document URI.

We have prototyped these techniques in a system that we call *similis*. We have evaluated our system on several tasks using a collection of graphs from the archive of the Swoogle Semantic Web search engine.

Text Based Similarity Metrics and Delta for Semantic Web Graphs

by

Krishnamurthy Koduvayur Viswanathan

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
MS Computer Science
2010

To my grandfather Kannan, who taught me my first alphabet

ACKNOWLEDGMENTS

I sincerely thank Dr. Tim Finin for guiding me through my research work. The Ebiquity Lab provided me with a great environment to spend countless days (and nights) working on this thesis. Thanks to Joshua Tauberer for creating SemWeb.NET; the best open source RDF library I have used. I would like to thank the creators of Swoogle for implementing the Swoogle Archive and Cache Services. It served to provide the test-data for my experiments, and I have no doubt that it will continue to be a rich source of data for future research too. I also thank the creators of Weka 3: Data Mining Software in Java, for providing an open source tool to build my classifiers.

Last but not the least, I would like to thank my family for the unending love and support they have given me over the years. My parents have always believed in me, which has been a constant source of motivation.

TABLE OF CONTENTS

DEDICATION . . . . **ii**

ACKNOWLEDGMENTS . . . . **iii**

LIST OF FIGURES . . . . **vii**

LIST OF TABLES . . . . **x**

Chapter 1 INTRODUCTION **1**

1.1 Motivation 1

1.2 Semantic Web Graph Similarity 4

Chapter 2 RELATED WORK **10**

2.1 Near Duplicate Detection and Schema Matching 10

2.2 Ontology and Concept Similarity 11

2.3 Ontology Versioning 12

2.4 Computing Change Between Semantic Web Documents 13

Chapter 3 SIMILARITY AND DELTA **15**

3.1 Problem Statement and Approach 15

3.1.1 Convert to n-triples 17

3.1.2 Canonical Representation 17

3.1.3 Reduced Forms 22

3.1.4 Similarity Metrics 26

3.1.5 Pairwise Comparison 30

3.1.6 Classification 30

3.2 Computing Delta Between Two Versions 33

3.2.1 Raw Delta 33

3.2.2 Delta After Deductive Closure 35

3.2.3 Delta at Class Level of an Ontology 35

3.2.4 Heuristic Method to Detect Class Renaming 38

Chapter 4 IMPLEMENTATION AND EVALUATION 39

4.1 Implementation 39

4.2 Construction of Data-set 39

4.3 Computation of Similarity Metrics 41

4.4 Detecting Similarity Between Pairs 41

4.4.1 Detecting Pairs that Differ only in Literal Content 41

4.4.2 Detecting Pairs that Differ Only in Base-URI 48

4.4.3 Detecting Pairs with a Versioning Relationship 53

4.5 Correctness of Delta Computation 55

Chapter 5 CONCLUSION AND FUTURE WORK 57

5.1 Future Directions 58

5.1.1 Scalability 58

5.1.2 Content of Delta Generated 58

5.1.3 Standard Ontology to Describe the Relationship Between Two SWDs 59

5.1.4 Predicting Direction of Change Between Two Versions 59

Appendix A APPENDIX 60

A.1 Sample Delta Outputs 60

REFERENCES 66

LIST OF FIGURES

1.1 Similar search results in google 2

1.2 Note the value of the attribute xml:base in the two ontology definitions. . . 5

1.3 Same RDF graph in two different formats: RDF/XML and n-triples 7

1.4 Search results for wine ontology on Swoogle 8

4.1 Swoogle Semantic Web Archive 40

4.2 Effect of CosineSim on structural similarity. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances. 45

4.3 Effect of LocalNameCosineSim on structural similarity. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances. 46

4.4 Effect of LocalNameNoLiteralJaccard on structural similarity. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances. 47

4.5 Effect of LocalNameNoLiteralSimhash on structural similartiy. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances. 48

4.6 Effect of LocalNameCosineSim on SWDs that differ only in base-URI. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances. 51

4.7 Effect of LocalNameNoLiteralContainment on SWDs that differ only in base-URI. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances. 52

4.8 Effect of OnlyLiteralJaccard on SWDs that differ only in base-URI. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances. 53

4.9 Each point on the horizontal axis represents a pair of SWDs which are snapshots of the same URL 54

A.1 Wine raw additive delta. Note the comment about the class renaming detected 61

A.2 Wine raw subtractive delta. Note the comment about the class renaming detected 62

A.3 Concept level delta (sauterne-additive) 63

A.4 Concept level delta (sauternes-subtractive) 64

A.5 Concept level delta (ChateauDYChemSauterne-subtractive) 65

A.6 Concept level delta (ChateauDYChemSauterne-additive) 65

LIST OF TABLES

1.1 Two semantic web documents with same structure but different literals . . . 6

3.1 Two semantic web documents before and after canonicalization 19

3.2 canonicalization process after steps 1-11 19

3.3 Non-distinctive triples 21

3.4 Functional and inverseFunctional properties can be used to deduce that two blank nodes are owl:sameAs or owl:differentThan. 22

3.5 Anatomy of an RDF triple 23

3.6 Semantic web document (input) to generate the reduced forms from 23

3.8 Only the literals are retained from the canonicalized n-triples graph 23

3.7 canonicalized n-triples format generated from the input document 24

3.9 All literals from the canonicalized n-triples form have been removed 24

3.10 Local name form of the canonicalized n-triples graph 25

3.11 Local name, no literal form of the canonicalized n-triples graph 25

3.12 Metrics computed between a pair of SWDs 31

3.13 A typical feature vector for a candidate pair 32

4.1 Stratified cross-validation (Naive Bayes) summary for similarity in classes and properties used 42

4.2 Detailed Accuracy by class (structural similarity), using Naive Bayes 42

4.3 Stratified cross-validation (Naive Bayes) summary for similarity in classes and properties used (including LocalNameCosineSim) 44

4.4 Detailed Accuracy by class (structural similarity - including feature LocalNameCosineSim) using Naive Bayes 44

4.5 Stratified cross-validation (SVM-linear-kernel) summary for similarity in classes and properties used 44

4.6 Detailed Accuracy by class (structural similarity), using SVM-linear kernel 44

4.7 Attribute relevance ranking using Infogain Attribute Evaluator and Ranker search for similarity in classes and properties used (top-six) 45

4.8 Stratified cross-validation (Naive Bayes) summary for pairs that differ only in Base URI 49

4.9 Detailed Accuracy by class (pairs different only in base URI), using Naive Bayes 49

4.10 Stratified cross-validation summary (SVM-linear kernel) for pairs that differ only in Base URI 50

4.11 Detailed Accuracy by class (pairs different only in base URI), using SVM - linear kernel 50

4.12 Evaluation on test set - summary (SVM-linear kernel) for pairs with a versioning relationship 55

4.13 Detailed Accuracy by class (versioning relationship), using SVM - linear kernel . . . 55

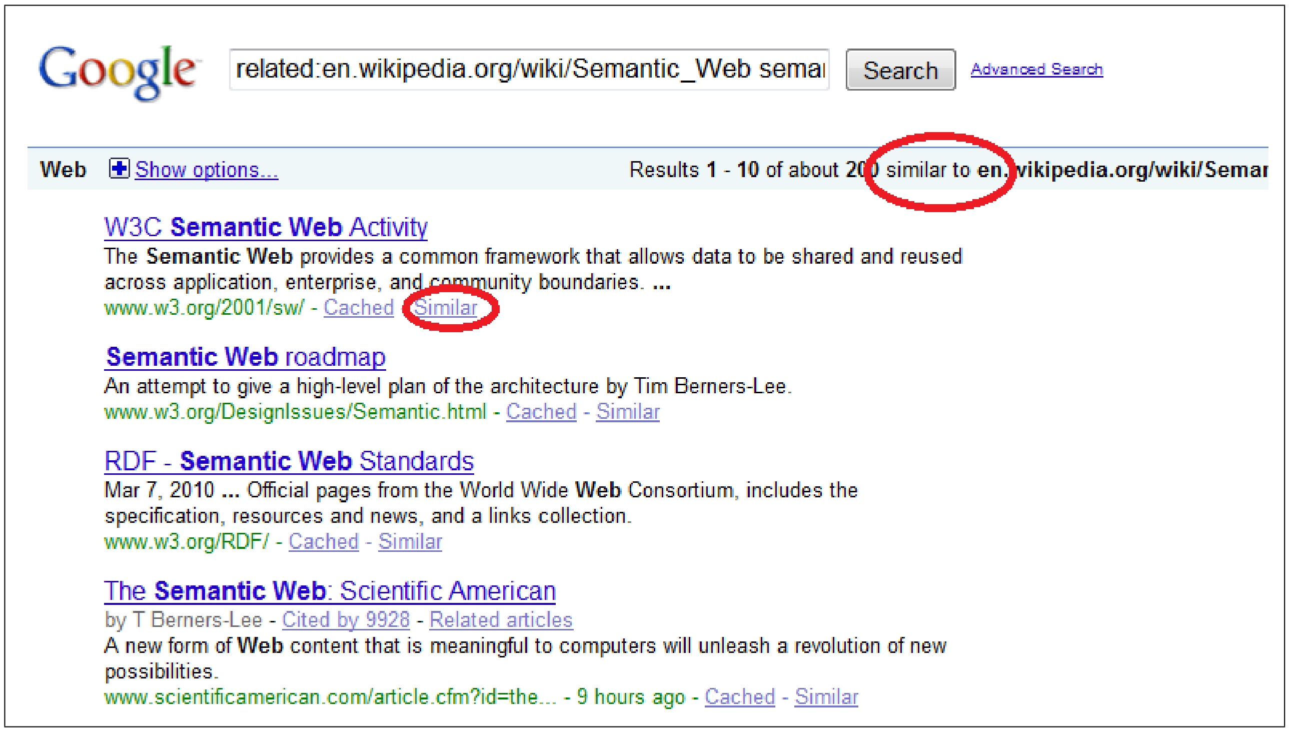
Chapter 1

INTRODUCTION

1.1 Motivation

It is estimated that almost 40% of web documents are duplicates of other pages (Manning, Raghavan, & Schütze 2008). It is desirable for a web crawler to be able to detect duplicates so that we can avoid crawling such pages. Duplicate and near duplicate pages increase the size of search engine indexes and reduce the quality of search results. The problem of near duplicate detection is well known in the field of information retrieval. Manku et al. (Manku, Jain, & Sarma 2007) present a good survey.

The same problem is being faced by Semantic Web search engines. Effective techniques are needed to determine similar semantic web documents on the web. Our work was motivated by the near duplicate detection studies in information retrieval. Everyday web search engines benefit from these techniques by being able to provide features like ”similar” results for every result generated in response to a search query. (see Fig. 1.4)



Results **1** - **10** of about 20 similar to **en.wikipedia.org/wiki/Sema**

W3C **Semantic Web** Activity
The **Semantic Web** provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. **...**
www.w3.org/2001/sw/ - Cached - Similar

Semantic Web roadmap
An attempt to give a high-level plan of the architecture by Tim Berners-Lee.
www.w3.org/DesignIssues/Semantic.html - Cached - Similar

RDF - **Semantic Web** Standards
Mar 7, 2010 **...** Official pages from the World Wide **Web** Consortium, includes the specification, resources and news, and a links collection.
www.w3.org/RDF/ - Cached - Similar

The **Semantic Web**: Scientific American
by T Berners-Lee - Cited by 9928 - Related articles
A new form of **Web** content that is meaningful to computers will unleash a revolution of new possibilities.
www.scientificamerican.com/article.cfm?id=the... - 9 hours ago - Cached - Similar

FIG. 1.1. Similar search results in google

We sought to use the techniques from this body of work, in the semantic web, thereby being able to provide similar functionalities for a semantic web search engine like Swoogle[1]. These techniques could also be used to compare the graphs returned by a SPARQL query run against a knowledge base on two different days.

The domain of semantic web documents, and more generally, that of semantic web graphs, is more complicated than that of documents based on plain text. In a text document, the order of the statements is essential to conveying meaning, whereas in a semantic web document, the statement ordering does not dictate the meaning of the content. As a result, equivalent semantic web documents may have completely different statement ordering.

[1]http://swoogle.umbc.edu/

Also, in text based near-duplicate detection, the meaning of the content is not a part of the problem, whereas in case of semantic web documents, the meaning plays a part in the problem. It is possible to have two different semantic web documents, which may become identical once we compute their deductive closure. In addition to statement ordering, we also need to contend with blank nodes in semantic web graphs. Blank nodes are anonymous resources that do not have a URI; and they are not literals.

We explore three different ways in which a pair of semantic web graphs can be similar to each other (identified in 1.2). We define text-based similarity metrics to characterize the relation between them. As a part of this, we identify whether they may be different versions of the same graph. Furthermore, if we determine a versioning relationship between a candidate pair, then we generate a delta between them.

These methods will enable a semantic web search engine to return links to documents that are similar to each search result, in response to a query. In addition, it will be able to generate in real time, a delta between the two results if there is a versioning relationship between them. We explain the meaning of ”similarity” and ”versioning” of semantic web graphs shortly.

Consider another use case for our study. There have been a lot of discussions on how linked data consumers could keep track of changes on a Linked Open Data (LOD) dataset [2] (dad). A service like pingthesemanticweb [3] allows publishers to register a newly created or modified document. The service then pushes a notification to interested consumers.

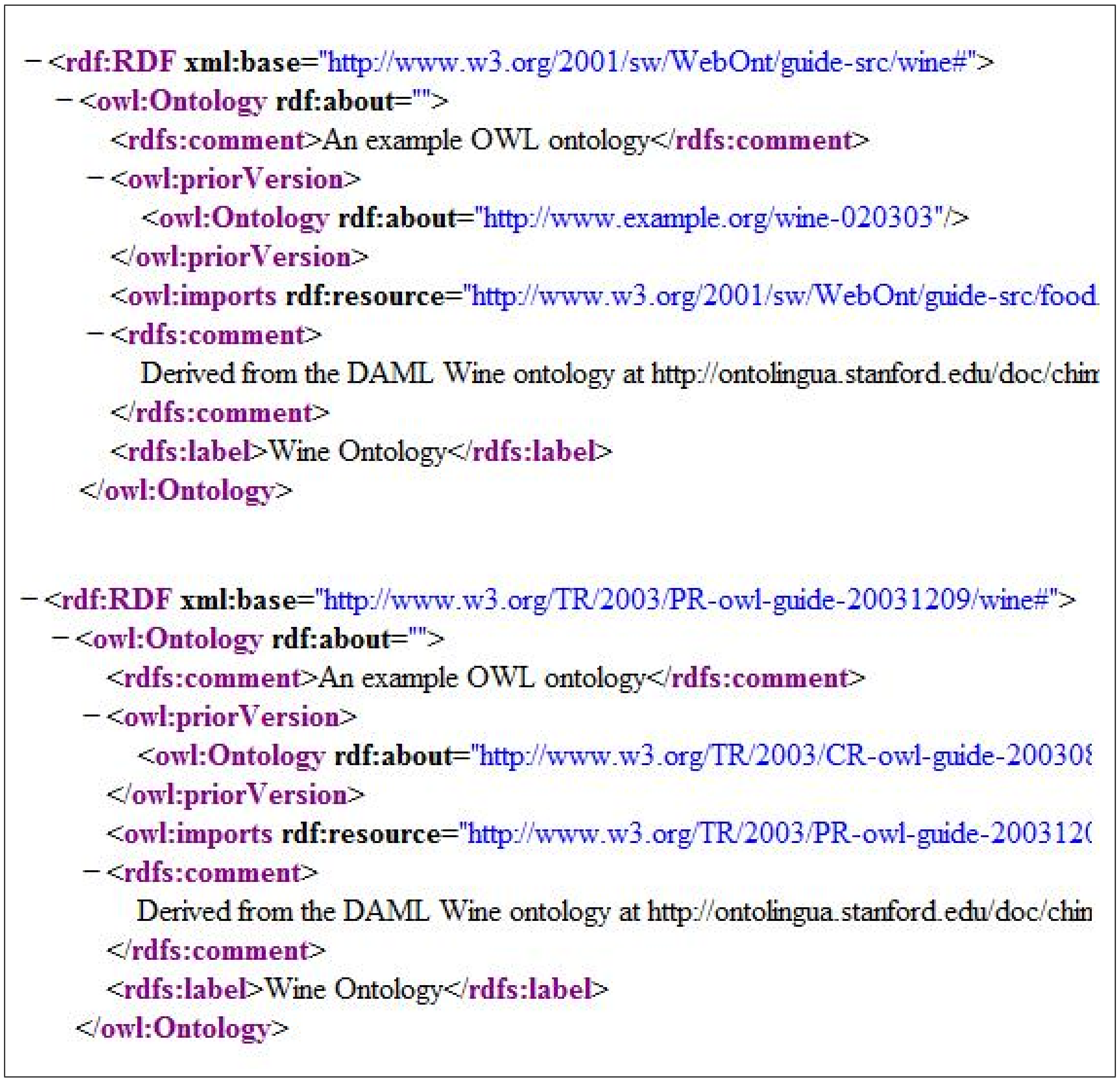
The problem with this model is that all publishers will have to register their changes with the service, but the advantage is that the updates are near real time. Imagine

[2]http://esw.w3.org/DatasetDynamics
[3]www.pingthesemanticweb.com

a semantic web search engine that can push updates to its consumers. The updates will be sent out when a new version of a document is discovered, along with a representation of the change between the two versions. These updates are not real time, but the advantage is that the changes will be discovered automatically by the search engine crawler instead of having each publisher register with the service. Updates can also be sent when a document similar to the one in which a consumer is interested in, is discovered. We use a collection of semantic web graphs from the archive of the Swoogle Semantic Web search engine to evaluate our approach on several tasks. In the following description, we use the terms ’semantic web document’ (SWD) and ’semantic web graph’ (SWG) interchangeably.

1.2 Semantic Web Graph Similarity

The archive or the Swoogle search engine (Ding *et al.* 2004) shows several examples of how ontologies and RDF documents evolve over time. For instance, there are several copies of the wine ontology (Smith, Welty, & McGuinness 2004) found on the web (developed by the W3C’s OWL Working Group, and used in the OWL Web Ontology Language Guide). Some of these are exact copies, while others use a different base URI. (see Figure 1.2)



```
– <rdf:RDF xml:base="http://www.w3.org/2001/sw/WebOnt/guide-src/wine#">
  – <owl:Ontology rdf:about="">
      <rdfs:comment>An example OWL ontology</rdfs:comment>
    – <owl:priorVersion>
        <owl:Ontology rdf:about="http://www.example.org/wine-020303"/>
      </owl:priorVersion>
      <owl:imports rdf:resource="http://www.w3.org/2001/sw/WebOnt/guide-src/food
    – <rdfs:comment>
        Derived from the DAML Wine ontology at http://ontolingua.stanford.edu/doc/chin
      </rdfs:comment>
      <rdfs:label>Wine Ontology</rdfs:label>
    </owl:Ontology>


– <rdf:RDF xml:base="http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#">
  – <owl:Ontology rdf:about="">
      <rdfs:comment>An example OWL ontology</rdfs:comment>
    – <owl:priorVersion>
        <owl:Ontology rdf:about="http://www.w3.org/TR/2003/CR-owl-guide-20030
      </owl:priorVersion>
      <owl:imports rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-200312
    – <rdfs:comment>
        Derived from the DAML Wine ontology at http://ontolingua.stanford.edu/doc/chin
      </rdfs:comment>
      <rdfs:label>Wine Ontology</rdfs:label>
    </owl:Ontology>
```

FIG. 1.2. Note the value of the attribute xml:base in the two ontology definitions.

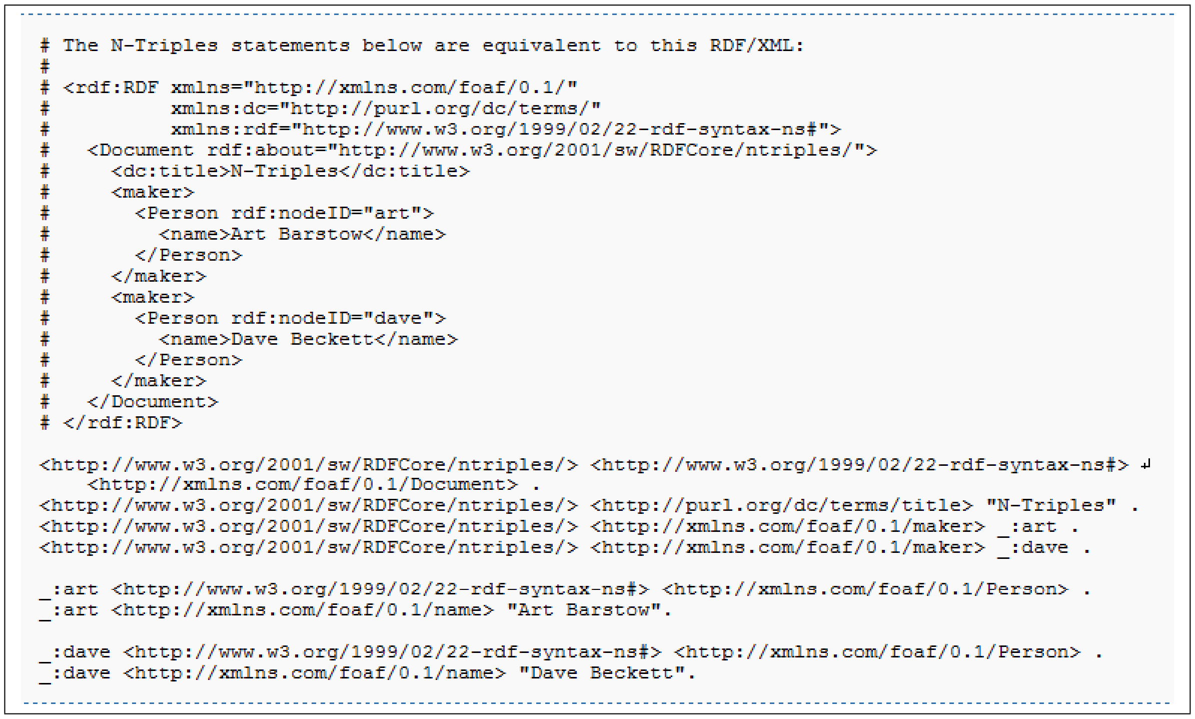
Some other versions contain small differences in the text content. We could also have semantic web documents (SWDs) that are similar in structure, but contain different instance data in terms of literals e.g. chat logs from forums or web services that generate FOAF (Brickley & Miller 2010) profiles from user entered information. (see Table 1.1)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<#JW>
a foaf:Person ;
foaf:name ”Jimmy Wales” ;
foaf:mbox <mailto:jwales@bomis.com> ;
foaf:homepage <http://www.jimmywales.com/> ;
foaf:nick ”Jimbo” ;
foaf:interest <http://www.wikimedia.org> ;
foaf:knows [
a foaf:Person ;
foaf:name ”Angela Beesley”
] .

<http://www.wikimedia.org>
rdfs:label ”Wikipedia” .
```

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<#JW>
a foaf:Person ;
foaf:name ”John Doe” ;
foaf:mbox <mailto:jdoe@example.com> ;
foaf:homepage <http://www.johndoe.com/> ;
foaf:nick ”John” ;
foaf:interest <http://www.wikimedia.org> ;
foaf:knows [
a foaf:Person ;
foaf:name ”Jane Walker”
] .

<http://www.wikimedia.org>
rdfs:label ”Wikipedia” .
```

Table 1.1. Two semantic web documents with same structure but different literals

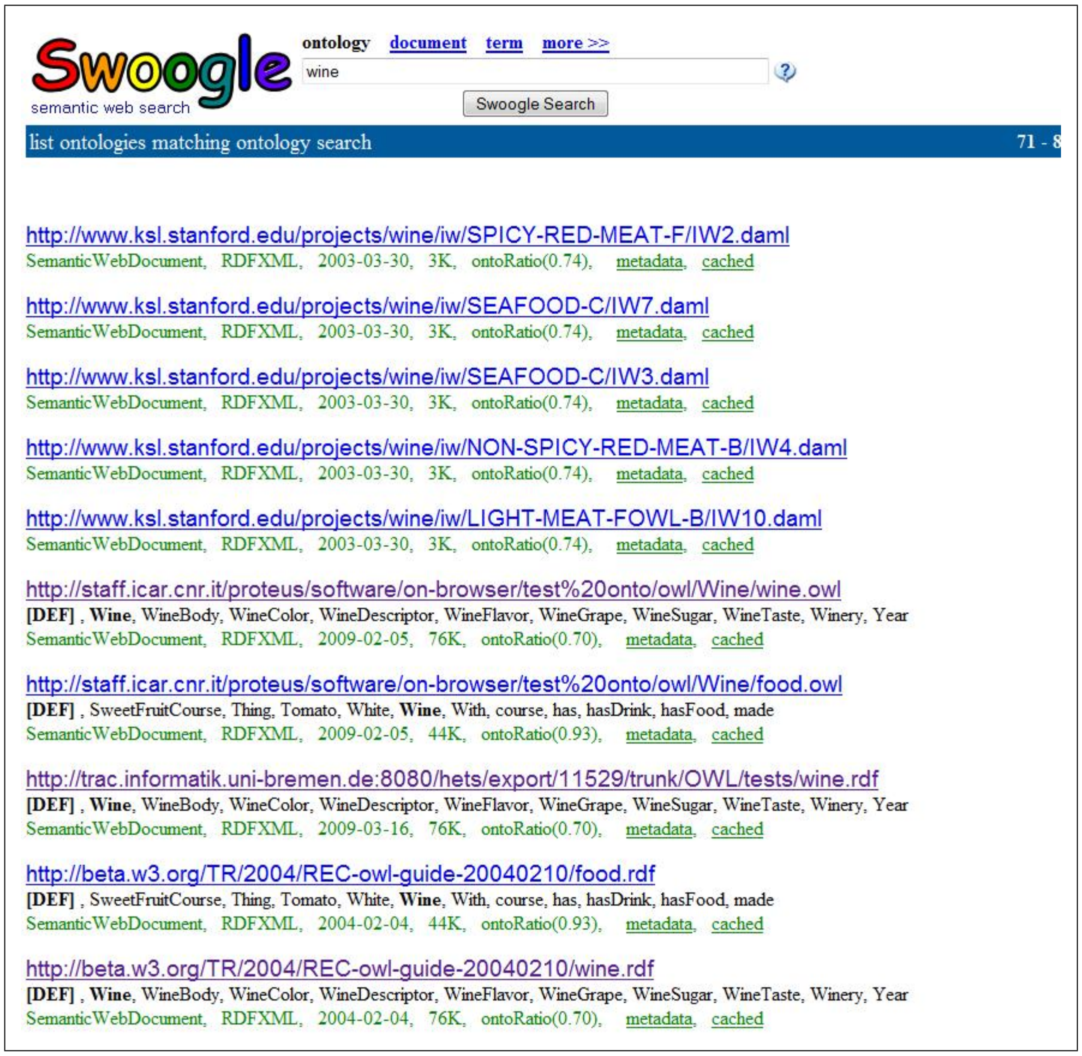
Hence similarity of two SWDs can be defined in terms of structure as well as content. We could have RDF documents that differ only in their serialization formats e.g. n-triples, N3, RDF/XML etc. Thus the same RDF document expressed in different serialization format, looks very different in text. (see Figure 1.3)



```
# The N-Triples statements below are equivalent to this RDF/XML:
#
# <rdf:RDF xmlns="http://xmlns.com/foaf/0.1/"
#          xmlns:dc="http://purl.org/dc/terms/"
#          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
#   <Document rdf:about="http://www.w3.org/2001/sw/RDFCore/ntriples/">
#     <dc:title>N-Triples</dc:title>
#     <maker>
#       <Person rdf:nodeID="art">
#         <name>Art Barstow</name>
#       </Person>
#     </maker>
#     <maker>
#       <Person rdf:nodeID="dave">
#         <name>Dave Beckett</name>
#       </Person>
#     </maker>
#   </Document>
# </rdf:RDF>

<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ↵
    <http://xmlns.com/foaf/0.1/Document> .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://purl.org/dc/terms/title> "N-Triples" .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://xmlns.com/foaf/0.1/maker> _:art .
<http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://xmlns.com/foaf/0.1/maker> _:dave .

_:art <http://www.w3.org/1999/02/22-rdf-syntax-ns#> <http://xmlns.com/foaf/0.1/Person> .
_:art <http://xmlns.com/foaf/0.1/name> "Art Barstow".

_:dave <http://www.w3.org/1999/02/22-rdf-syntax-ns#> <http://xmlns.com/foaf/0.1/Person> .
_:dave <http://xmlns.com/foaf/0.1/name> "Dave Beckett".
```

FIG. 1.3. Same RDF graph in two different formats: RDF/XML and n-triples

As an example, searching for "wine" on swoogle returns several results including 14 different near-duplicate copies of the wine ontology. Amongst themselves, they represent, two different versions of the ontology. We observed three different base-URIs amongst these, distributed into groups of three, three and eight results.



wine

list ontologies matching ontology search 71 - 8

http://www.ksl.stanford.edu/projects/wine/iw/SPICY-RED-MEAT-F/IW2.daml
SemanticWebDocument, RDFXML, 2003-03-30, 3K, ontoRatio(0.74), metadata, cached

http://www.ksl.stanford.edu/projects/wine/iw/SEAFOOD-C/IW7.daml
SemanticWebDocument, RDFXML, 2003-03-30, 3K, ontoRatio(0.74), metadata, cached

http://www.ksl.stanford.edu/projects/wine/iw/SEAFOOD-C/IW3.daml
SemanticWebDocument, RDFXML, 2003-03-30, 3K, ontoRatio(0.74), metadata, cached

http://www.ksl.stanford.edu/projects/wine/iw/NON-SPICY-RED-MEAT-B/IW4.daml
SemanticWebDocument, RDFXML, 2003-03-30, 3K, ontoRatio(0.74), metadata, cached

http://www.ksl.stanford.edu/projects/wine/iw/LIGHT-MEAT-FOWL-B/IW10.daml
SemanticWebDocument, RDFXML, 2003-03-30, 3K, ontoRatio(0.74), metadata, cached

http://staff.icar.cnr.it/proteus/software/on-browser/test%20onto/owl/Wine/wine.owl
[DEF] , **Wine**, WineBody, WineColor, WineDescriptor, WineFlavor, WineGrape, WineSugar, WineTaste, Winery, Year
SemanticWebDocument, RDFXML, 2009-02-05, 76K, ontoRatio(0.70), metadata, cached

http://staff.icar.cnr.it/proteus/software/on-browser/test%20onto/owl/Wine/food.owl
[DEF] , SweetFruitCourse, Thing, Tomato, White, **Wine**, With, course, has, hasDrink, hasFood, made
SemanticWebDocument, RDFXML, 2009-02-05, 44K, ontoRatio(0.93), metadata, cached

http://trac.informatik.uni-bremen.de:8080/hets/export/11529/trunk/OWL/tests/wine.rdf
[DEF] , **Wine**, WineBody, WineColor, WineDescriptor, WineFlavor, WineGrape, WineSugar, WineTaste, Winery, Year
SemanticWebDocument, RDFXML, 2009-03-16, 76K, ontoRatio(0.70), metadata, cached

http://beta.w3.org/TR/2004/REC-owl-guide-20040210/food.rdf
[DEF] , SweetFruitCourse, Thing, Tomato, White, **Wine**, With, course, has, hasDrink, hasFood, made
SemanticWebDocument, RDFXML, 2004-02-04, 44K, ontoRatio(0.93), metadata, cached

http://beta.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf
[DEF] , **Wine**, WineBody, WineColor, WineDescriptor, WineFlavor, WineGrape, WineSugar, WineTaste, Winery, Year
SemanticWebDocument, RDFXML, 2004-02-04, 76K, ontoRatio(0.70), metadata, cached

FIG. 1.4. Search results for wine ontology on Swoogle

Our study can also be applied to aid the detection of co-referent FOAF profiles. FOAF stands for Friend of a Friend, which is a machine readable ontology that describes people and their relation to other people. The FOAF profile of a person does not have a unique ID. As a result of this, it is possible for a person to have multiple versions of FOAF profiles online. Sleeman et al. (Sleeman & Finin 2010) have proposed a machine learning based approach to solve this problem.

Chapter 2

RELATED WORK

Similarity detection is used in various fields of computer science such as information retrieval, databases, and semantic web, amongst others. The field of IR has used near-duplicate-detection techniques for purposes such as detecting web mirrors, clustering related documents, data-extraction, plagiarism detection, spam detection, and finding duplicates in domain specific corpora. Database researchers have been interested in database schema matching, which is the process of identifying whether two database objects are semantically related. (Rahm & Bernstein 2001) present a good survey. A related idea in the field of semantic web is called ontology alignment or ontology matching which refers to the process of determining correspondeces between ontology concepts.

2.1 Near Duplicate Detection and Schema Matching

We attempt to identify near duplicate semantic web documents that have been created due to revisions, modifications, splitting, copying and merging of documents in a domain specific corpus. Similar studies have been done in the past on text documents. Conrad and Schriber (Conrad & Schriber 2004) describe the creation of a test for inexact duplicate detection in legal documents. This paper is a good example of how near duplicate detection is done in a domain specific environment. They describe domain specific parameters for calling a pair of documents as near duplicates of each other. Manku et al. (Manku, Jain, & Sarma 2007) describe the design of a near duplicate detection system on a multi-billion page repository.

Udi Manber (Manber 1994) describes a system called *sif* that harnesses a preprocessed index to find files in a filesystem that are similar to each other. (Conrad & Schriber 2004) and (Manber 1994) provide good use-cases that are relevant to their respective domains. Paes Leme et al. (Leme *et al.* 2008) use semantic web technologies like the RDF serialization to perform schema matching for databases. They describe the schemas in RDF, containing only class and property definitions using simple XML schema types. Then the two schemas are matched by matching the classes and properties defined for both the schemas. Traditional similarity metrics from the literature are used for this purpose.

2.2 Ontology and Concept Similarity

There has been some work on the problem of detecting or calculating similarity between semantic web graphs, but the applications and use-cases have been different from ours. Radoslaw et al. (Radoslaw Oldakowski 2005) describe a framework for computing the similarity between two arbitrary objects specified as RDF graphs based on similarities of specified object properties. This method requires human configuration for every pair of graphs to be compared. Hau et al. (Hau, Lee, & Darlington 2005) describe a semantic similarity measure for web service descriptions in OWL-S. This method is based on measuring the common information between two objects. This method is specific to particular ontologies. The authors describe it with respect to OWL Lite constructs.

Bunke and Shearer (Bunke & Shearer 1998) present a graph based distance measure that is based on the maximal common subgraph of two graphs. This method may be suitable for computing the similarity between two RDF graphs, but in general, the algorithms for computing the maximal common subgraph of two graphs are computationally very expensive (Levi 1973). These methods are not suitable for our use cases. Carroll (Carroll 2002) explores the equality of two RDF graphs in the light of the graph isomorphism literature. Maedche and Staab (Maedche & Staab 2002) present a concept similarity model based on ontology terminology and other structural aspects of the ontology. However, this paper does not evaluate the ontology similarity process. Ehring et al. (Ehrig *et al.* 2005) describe a framework that aims at comparing concepts across ontologies, and not ontologies themselves. David and Euzenat (David & Euzenat 2008) present a number of distance measures for ontology matching and state that simple measures like cosine similarity on a term-frequency vector give accurate results. This is one of the measures we use in our study.

2.3 Ontology Versioning

There are theoretical and experimental studies which deal with ontology versioning. Heflin et al. (Heflin & Hendler 2000) describe a theoretical model for managing multiple versions of ontologies in a distributed environment. Klein et al. (Klein & Fensel 2001) define ontology versioning as *the ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology*. Tools such as Evolva (Zablith 2009) were created to help ontology developers manage ontology versions locally. However such tools use different ways to encode information which is not transferred when ontologies are collected and distributed through online repositories. The OWL standard (Smith, Welty, & McGuinness 2004) includes a primitive *owl:priorVersion* to be used by ontology developers to indicate a versioning relationship with a prior version of the same ontology. However, primitives such as this are rarely used. A search on the Swoogle search engine for instances of this *owl:priorVersion* returns a sum total of nine results (as on June

15, 2010).

Allocca et al. (Allocca, d'Aquin, & Motta 2009) describe an approach to identify ontology versioning relations by looking for common patterns in the URIs, but ignore the content of the ontology. Such heuristic based methods cannot identify a version of the ontology that may have a different base URI, because it may be on a different server.

2.4 Computing Change Between Semantic Web Documents

With respect to determining the change between two semantic web documents, Papavassiliou et al (Papavassiliou *et al.* 2009) propose a change detection language, framework and algorithm. They define a mathematical model for change between two RDF/S KBs. They enlist the low level changes between the two KBs and map these to higher level conceptual changes based on detailed rules that they have proposed. There are no experimental results related to this work. Zeginis et al. (Zeginis, Tzitzikas, & Christophides 2007) define the type of change operations that are normally used while comparing two RDF models, in addition to the semantics of application of these change operations. They propose the use of a new change operation. This is also a theoretical work without any experimental results. Tools like PromptDiff (Noy *et al.* 2002) generate the differences between two versions of the same ontology. It uses a variety of heuristic matchers along with a fixed point algorithm to apply all of the matchers. Klein et al. (Klein *et al.* 2002) consider each ontology as an XML document. The document is split into groups of top level XML elements, each of which identify a specific concept that has been defined. Each of these groups are then serialized into the n-triple format, and then these triples are compared between the two versions of the ontology according to a set of rules specific to the vocabulary in which the ontology has been defined. The Graph Update Ontology (guo) is an ongoing effort at creating an ontology to describe an RDF based diff for RDF graphs.

Thus a lot of past work has focused on comparing ontology versions. We need methods that could be applied to any semantic web document in general. Our goal is to develop a technique that can automatically generate the difference description between candidate documents pairs in a domain specific corpus. This would require both automatic detection of similarity, and generation of a diff. Such a functionality would enable a semantic web search engine to provide related documents for each search result generated. Each related document would also be associated with a diff that would describe how this document is different from the one contained in the result.

Chapter 3

SIMILARITY AND DELTA

3.1 Problem Statement and Approach

RDF Document or Semantic Web Document: It is the serialization of an RDF graph, and may be expressed in any of the following serialization formats: RDF/XML-abbrev, n-triples, n3. Usually, these documents have suffixes '.rdf', '.nt', and '.n3' respectively. We do not consider an XHTML document with embedded RDF content to be a semantic web document.

Ontology Ratio: Swoogle[1] defines the proportion of defined classes and properties to all the terms defined in the SWD. Given a SWD foo, its ontology ratio R(foo) is calculated by the following equation.

$$R(foo) = (\|C(foo)\| + \|P(foo)\|)/(\|C(foo)\| + \|P(foo)\| + \|I(foo)\|)$$

where C(foo), P(foo), I(foo) refer to the sets of defined classes, defined properties and defined individuals in foo respectively.

Semantic Web Ontology: It is a special type of SWD that defines classes and properties that may be used by the same or other SWDs. Swoogle defines a semantic web ontology as a SWD whose Ontology Ratio is no less than 0.8.

[1]http://swoogle.umbc.edu/index.php?option=com_swoogle_manual&manual=glossary

Semantic Web Data File: It is a SWD that does not define any classes or properties.

Versioning Relationship: Klein et al. (Klein & Fensel 2001) define ontology versioning as *"the ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology"*. Allocca et al. (Allocca, d'Aquin, & Motta 2009) state that *"ontology versioning means that there are multiple variants of an ontology around and that these variants should be managed and monitored"*. Similarly, we extend this definition to semantic web documents in general. We define two semantic web documents as having a versioning relationship if they are variants of the same semantic web graph. The web is dynamic. New content gets added everyday, and existing content keeps getting modified. The content of some URLs changes rapidly, while others change relatively slowly or even remain static. Some of these changes do not affect the semantic content e.g. spelling corrections, punctuations, reorganization of paragraphs etc. Other modifications may change the semantics of the content or even delete information. Thus variants of a semantic web document on the web are created due to the dynamic nature of the web.

Problem1 Definition: *Given a collection of semantic web graphs in the form of RDF documents, identify pairs of graphs that may be similar to each other. Characterize the similarity into one or more of the three cases:*

1. Same classes and properties used, but differ only in the literal content
2. Differ only in the base-URI used
3. Are different versions of the same graph i.e. have a versioning relationship

Problem2 Definition: *Generate a delta between pairs that have been identified as having a versioning relationship between them.*

Our input corpus is in the form of a set of RDF documents. We compute a number of similarity scores to determine the type and extent of similarity between them. Similarity between two RDF graphs may be defined in terms of namespaces used in the graphs, classes and properties used, local names of classes and properties used, all the triples occurring in the graph, all the triples including the inferred triples in the graph etc. This method may be used for ontologies as well as RDF data files. Our approach involves the following steps:

3.1.1 Convert to n-triples

Two semantically identical RDF graphs can be textually different if their serialization format is different (RDF/XML, n-triples, n3 etc.). Text based similarity functions are sensitive to the serialization format and may report two similar documents as different if they differ in the serialization format. Hence we convert all the documents into a uniform serialization format before comparing them.

3.1.2 Canonical Representation

Text based comparison methods are also affected by blank node identifiers and statement ordering. Semantic web graphs may contain any number of blank nodes. A blank node has an ID whose scope is within the particular graph; and it can be differentiated from another blank node within the same graph, but not within another graph.

Consider two equivalent semantic web graphs (document1.nt, and document2.nt) that use different blank node IDs (see Table 3.1). Both the graphs are semantically equivalent, but have different representations in text.

Algorithm 1 assigns consistent IDs to blank nodes and orders the statements lexicographically. It removes non-determinism such as random blank-node IDs and variable statement ordering and transforms two semantically equivalent graphs into the same canonical representation. This algorithm is based on Jeremy Carroll's canonicalization algorithm (Carroll 2003).

Algorithm 1 Convert SWD into a canonical representation

```
1: Load all triples in the graph into memory
2: for all $triple$ in graph do
3:    if $triple.subject$ is a BNode then
4:       $triple.subject \leftarrow$ " $\sim$ "
5:       $triple.meta.subjectcomment \leftarrow triple.subject.nodeID$
6:    end if
7:    if $triple.object$ is a BNode then
8:       $triple.object \leftarrow$ " $\sim$ "
9:       $triple.meta.objectcomment \leftarrow triple.object.nodeID$
10:   end if
11: end for
12: Sort all the triples in alphabetical order. (Key for the sort is the text representation of
    the triple. Ignore the metadata comments)
13: $bTable \leftarrow$ new HashTable
14: $newID \leftarrow \_ : g1$
15: for all $triple$ in sorted graph do
16:    if $triple.meta.objectcomment$ is not empty then
17:       if $bTable$ not containsKey($triple.meta.objectcomment$) then
18:          $bTable[triple.meta.objectcomment] \leftarrow$ new BNode(newID)
19:          increment $newID$
20:       end if
21:       $triple.meta.object \leftarrow bTable[triple.meta.objectcomment]$
22:       delete $triple.meta.objectcomment$
23:    end if
24:    if $triple.meta.subjectcomment$ is not empty then
25:       if $bTable$ not containsKey($triple.meta.subjectcomment$) then
26:          $bTable[triple.meta.subjectcomment] \leftarrow$ new BNode($newID$)
27:          increment $newID$
28:       end if
29:       $triple.meta.subject \leftarrow bTable[triple.meta.subjectcomment]$
30:       delete $triple.meta.subjectcomment$
31:    end if
32: end for
33: Sort all the triples in alphabetical order (Key for the sort is the text representation of
    the triple. Ignore the metadata comments)
```

document1.nt (input)	**canonicalized document1.nt (output)**
<person:John> <a:livesIn> _:x .	_:g2 <a:hasCapital> _:g1 .
_:x <a:IsPartOf> ”USA” .	_:g2 <a:IsPartOf> ”USA” .
<person:John> <a:likes> ”cheese” .	<person:John> <a:likes> ”cheese” .
_:x <a:hasCapital> _:y .	<person:John> <a:livesIn> _:g2 .
document2.nt (input)	**canonicalized document2.nt (output)**
_:a <a:hasCapital> _:b .	_:g2 <a:hasCapital> _:g1 .
<person:John> <a:livesIn> _:a .	_:g2 <a:IsPartOf> ”USA” .
_:a <a:IsPartOf> ”USA” .	<person:John> <a:likes> ”cheese” .
<person:John> <a:likes> ”cheese” .	<person:John> <a:livesIn> _:g2 .

Table 3.1. Two semantic web documents before and after canonicalization

” ∼ ” <a:hasCapital> ” ∼ ” . #_:x _:y
” ∼ ” <a:IsPartOf> ”USA” . #_:x
<person:John> <a:likes> ”cheese” .
<person:John> <a:livesIn> ” ∼ ” . #_:x

Table 3.2. canonicalization process after steps 1-11

Description of the algorithm Steps 1-11 serially traverse through each triple in the graph. If the subject or object of a statement is a blank node, then it is replaced by a tilde character. The tilde character is an escape character that does not naturally occur in a triple. Next the triples in the graph are sorted in lexicographic order by treating the string representation of the triple as the key. This sorting process ignores the meta-data comments created in steps 1-11. The output after this stage is as shown in Table 3.2

Steps 15-23 traverse through each statement in the sorted graph. If a statement has a subject-comment, then it is looked up in the blank-node table. If an entry does not exist for this node, then a new entry is created in the table, with the subject-comment as key, and a new blank node identifier as value. The blank node identifier is then incremented to be used again. In Lines 21-22 the subject of the statement is replaced by the entry from the table, and the subject-comment is deleted. Similarly, lines 24-31 do the same processing for objects in statements. Finally the triples are sorted once again so that they are in lexicographic order. The output is as seen in Table 3.1.

As can be seen from Table 3.1, the two input documents have the same content, but differ only in the order of the triples and the blank node identifiers used. After running the canonicalization algorithm, both the documents become identical. We use this method to convert every input document to its canonical representation, before we process it further for similarity detection. The graphs are thus transformed into a deterministic serialization format. Note that the RDF specification does not allow predicates to be blank nodes.

Limitations of the Algorithm: Non-distinctive triples The algorithm is able to correctly rename blank nodes in only those triples that can be identified as unique in the graph even after all blank nodes are treated as equal. The two graphs shown Table 3.3 are equivalent to each other, but when the canonicalization algorithm is applied on them, they may not get the same blank node identifiers. When the canonicalization algorithm replaces the blank nodes by tildes, then some adjacent triples in a single graph become indistinguishable from each other (as seen in Table 3.3). Such triples are called non-distinctive triples.

input graphs	**after removing blank nodes**
_:b1 :foo _:b2 .	$\sim$:foo $\sim$. #b1 b2
_:b2 :foo _:b3 .	$\sim$:foo $\sim$. #b2 b3
_:b1 :foo _:b3 .	$\sim$:foo $\sim$. #b1 b3
_:b1 :bar ”a” .	$\sim$:bar ”a” . #b1
_:b1 :foo _:b3 .	$\sim$:foo $\sim$. #b1 b3
_:b2 :foo _:b3 .	$\sim$:foo $\sim$. #b2 b3
_:b1 :foo _:b2 .	$\sim$:foo $\sim$. #b1 b2
_:b1 :bar ”a” .	$\sim$:bar ”a” . #b1

Table 3.3. Non-distinctive triples

The problem is that for a group of n non-distinct triples, there are $n!$ ways of renaming the blank nodes, and there may be several such groups. Thus, for graphs with non-distinctive triples, a single unique canonical form does not exist. In order to compare such graphs, we would have to compare each of the possible canonical forms for both graphs. If we have one graph having k groups of n non-distinctive triples, and another graph having l groups of m non-distinctive triples, then the total number of graph comparisons would be $\Theta(m!n!)$. Our system avoids having to deal with this combinatorial explosion by throwing an exception when such a case is encountered.

graph 1	graph 2
_:b1 ex:fp ex:o1	_:g8 ex:fp ex:o1
_:b2 ex:ifp ex:o2	_:g7 ex:ifp ex:o2
_:b3 ex:ifp1 ex:o3	_:g4 ex:ifp1 ex:o4

Table 3.4. Functional and inverseFunctional properties can be used to deduce that two blank nodes are owl:sameAs or owl:differentThan.

Consider the example in Figure 3.4. In this case, we have functional and inverse functional properties in the triples. In such cases, it may be possible to determine whether two blank nodes refer to teh same or distinct objects.

Assume that *ex:fp* is an *owl:functionalProperty* and *ex:ifp* is an *owl:inverseFunctionalProperty*. We can conclude that *_:b1 owl:sameAs _:g8* and *_:b2 owl:sameAs _:g7*. If we know that a property is function and its inverse is inverseFunctional, then we can also deduce that two blank nodes are distinct. A real world example might be an identification number that is assigned to one and only one person. If *ex:ifp1* is such a property and we can prove that e*ex:03* and *ex:o4* are distinct (e.g., if they are different literals), then we can conclude that *:_b3 owl:differentThan _:g4*. It is also possible to determine that two blank nodes are in fact different by making use of cardiniality constraints.

3.1.3 Reduced Forms

We discussed the different forms of similarity that is possible between two candidate graphs. In order to detect the various forms, the original graphs will have to be decomposed into forms where these can be detected. In each of our reduced forms, we transform each triple[2] in the graph.

[2]see Table 3.5

Entity URI	<http://www.w3.org/2001/sw/guide-src/wine#hasSugar>
Base URI	<http://www.w3.org/2001/sw/guide-src/wine>
Local Name	<wine>

Table 3.5. Anatomy of an RDF triple

Input semantic web document
_:blank1 <http://purl.org/dc/elements/1.1/title> ”Unwritten work” .
_:blank2 <http://purl.org/dc/elements/1.1/title> ”The League of Procrastinators” .
_:blank1 <http://purl.org/dc/elements/1.1/creator> _:blank2 .
_:blank1 <http://purl.org/dc/elements/1.1/contributor> <http://put-off.org> .

Table 3.6. Semantic web document (input) to generate the reduced forms from

We generate the following reduced forms from the canonicalized n-triples form (Table 3.7) of each graph:

1. A document that contains only the literals from the canonicalized n-triples file. This reduced form lets us compare only the textual content within a graph, separated from the rest of the graph. Note that type information of literals is retained. (Table 3.8)

Only-Literals format
”Unwritten work”
”The League of Procrastinators”

Table 3.8. Only the literals are retained from the canonicalized n-triples graph

2. A document that has all the literals replaced by the empty string [3] . This reduced form lets us compare only the classes and properties used, regardless of the literal content. (Table 3.9). It is not essential to keep the empty strings (””) in the reduced form. These are kept only for the sake of readability. The functioning of the system would be unaffected even if these empty strings are not retained in the reduced form(s).

[3]Another viable approach is to replace each literal string by its XSD data type

canonicalized n-triples format
_:g1 <http://purl.org/dc/elements/1.1/contributor> <http://put-off.org> .
_:g1 <http://purl.org/dc/elements/1.1/creator> _:g2 .
_:g1 <http://purl.org/dc/elements/1.1/title> ”Unwritten work” .
_:g2 <http://purl.org/dc/elements/1.1/title> ”The League of Procrastinators” .

Table 3.7. canonicalized n-triples format generated from the input document

No-Literals format
_:g1 <http://purl.org/dc/elements/1.1/contributor> <http://put-off.org> .
_:g1 <http://purl.org/dc/elements/1.1/creator> _:g2 .
_:g1 <http://purl.org/dc/elements/1.1/title> ”” .
_:g2 <http://purl.org/dc/elements/1.1/title> ”” .

Table 3.9. All literals from the canonicalized n-triples form have been removed

3. A document that has the base-URI of every node replaced by the empty string i.e. the URI of each entity is reduced to its local name only. This reduced form lets us compare only the local names of the classes and properties used in both the graphs. This form does contain the literal content of the graph. (Table 3.10).

   Consider two triples having the same subject and predicate, but different blank nodes as objects. We keep the blank nodes in all reduced forms so that the triples can be differentiated from each other.

Local-name format
_:g1 <contributor> <put-off.org> .
_:g1 <creator> _:g2 .
_:g1 <title> ”Unwritten work” .
_:g2 <title> ”The League of Procrastinators” .

Table 3.10. Local name form of the canonicalized n-triples graph

4. A document that has all the literals and the base-URI of every node replaced by the empty string. This reduced form is a combination of reduced forms two and three. It has only the local names of classes and properties used, and all the literal content is removed. This form is used to do a comparison of two sematic web graphs with respect to their non-literal content. (Table 3.11)

Local-Name-No-Literals format
_:g1 <contributor> <put-off.org> .
_:g1 <creator> _:g2 .
_:g1 <title> ”” .
_:g2 <title> ”” .

Table 3.11. Local name, no literal form of the canonicalized n-triples graph

Thus, each semantic web graph has a canonical representation, and four reduced forms i.e. five forms in all.

3.1.4 Similarity Metrics

In our study, all of the semantic web graphs are serialized as documents which are canonicalized and converted into reduced forms. This allows us to use text based similarity measures that are used in the field of information retrieval. Different kinds of metrics are defined depending on the kind of similarity we are trying to measure, such as structural similarity with respect to classes and properties used, or textual content similarity, or both.

A similarity metric between a pair of semantic web documents (SWDs) from a collection G is a function $s : G \times G \rightarrow \mathbb{R}$. Note that these measures are computed over the reduced forms of each semantic web graph as described above. Following are some of the properties of similarity measures:

1. **Non negativity** $\forall\, G_1, G_2 \in G, 0 \leq s(G_1, G_2)$
2. **Identity of Indiscernibles** $\forall\, G_1, G_2 \in G$, $s(G_1, G_2) = 1$ if and only if $G_1 \equiv G_2$, and as a corollary, $s(G_1, G_1) = 1$
3. **Symmetry** $\forall\, G_1, G_2 \in G$, $s(G_1, G_2) = s(G_2, G_1)$
4. **Normality** Those metrics are said to be normal whose value ranges in the interval [0,1]

The value of a similarity metric between two SWDs ranges between 0 and 1. If SWDs are very similar to each other, the value of the metric is closer to 1; whereas metrics for graphs that are very dissimilar will be closer to 0. In particular, we use the following measures (the exact procedure of computing pairwise metrics using reduced forms is explained shortly):

Jaccard similarity and containment : The well known Jaccard similarity metric measures the overlap between two sets. The Jaccard similarity between two sets A and B is defined as:

$$jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

This quantity ranges between 0 (completely different) and 1 (identical).

Containment of set A in set B is defined as:

$$containment(A, B) = \frac{|A \cap B|}{|A|} \tag{3.2}$$

It should be noted that the containment measures is not a similarity measure, since it is not symmetric.

We first extract character 4-grams from each document and construct sets of character 4-grams. These sets are then used to compute the measures as indicated above. The 4-grams are computed by running a window four characters wide over the text representation.

Whereas the Jaccard measure indicates the similarity between two sets of n-grams, the containment measure indicates whether one set of n-grams is contained within the other: 0 indicating no containment and 1 indicating complete containment. We use this containment measure to determine whether one semantic web graph is an older/newer version of a similar graph. A high value for both Jaccard and containment metrics indicates a strong possibility of a versioning or equivalence relation between two.

Cosine Similarity between semantic features : Each semantic web document is represented as a vector of terms. These terms are the subject, predicate, and object of the triples appearing in the semantic web document. Consider G is the set of SWDs and Γ is the set of terms appearing in these SWDs, then a semantic web document vector containing the terms $\Gamma_j = (t_1, t_2, ..., t_n)$ is defined as $\overrightarrow{V_j} = (\gamma_1 t_1, \gamma_2 t_2, ... \gamma_n t_n)$ where each γ_i represents the weight of the term $t_i \in \Gamma_j$.

To construct this vector, the non-blank, non-literal nodes from each SWD are extracted and their term-frequency (TF) in the SWD is used as the feature weight. Two vectors are generated for each SWD: one using the terms as the features, and another using only the local-names of the terms (i.e. ignoring the base-URI). The cosine similarity between two vectors is defined as:

$$similarity(A, B) = cos(\theta_{A,B}) = \frac{\overrightarrow{A}.\overrightarrow{B}}{\|\overrightarrow{A}\|\|\overrightarrow{B}\|} \tag{3.3}$$

The cosine similarity metric indicates similarity in classes and properties used.

Charikar's Simhash : This fingerprinting technique was developed by Charikar (Charikar 2002). It is a Locality Sensitive Hashing (Indyk & Motwani 1998) method which has a property that such fingerprints of similar documents differ in a small number of bit positions. Simhash fingerprints are like regular hashes e.g. MD5, but with the additional property that simhash fingerprints of similar documents are mapped together. Hence in order to find whether two documents are similar to each other, we simply compute their simhashes and determine the Hamming distance between them. If the Hamming distance is smaller than a predetermined threshold, then we can conclude that the two documents are similar to each other.

Note that the hamming distance measure is in fact a distance metric, instead of a similarity metric. A distance metric is a function $d : G \times G \rightarrow \mathbb{R}$ which defines the distance between two elements in a set, and has the following properties:

1. **Non negativity** $\forall\, G_1, G_2 \in G$, $0 \leq d(G_1, G_2)$
2. **Identity of Indiscernibles** $\forall\, G_1, G_2 \in G$, $d(G_1, G_2) = 0$ if and only if $G_1 \equiv G_2$, and as a corollary, $d(G_1, G_1) = 0$

3. **Symmetry** $\forall\, G_1, G_2 \in G$, $d(G_1, G_2) = d(G_2, G_1)$

4. **Triangle Inequality** $\forall\, G_1, G_2, G_3 \in G$, $d(G_1, G_3) \leq d(G_1, G_2) + d(G_2, G_3)$

The basic idea behind the method is to compute 'sketches' of original documents such that the similarity between two sketches can provide an indication of the similarity between the original documents. The method essentially maps an n-dimensional feature vector into a k-dimensional bit vector, where $n >> k$. Our implementation of a 128-bit simhash is as follows:

1. Tokenize the document into character 3-grams.

2. Compute the hash h_i of each token t_i using a traditional hash function.

3. An integer vector v of length 128 is initialized to the zero vector.

4. For each h_i, update v as follows:

   (a) **if** $h_i[j] == 1$ **then** $v[i] = v[i] + 1$

   (b) **else** $v[i] = v[i] - 1$

5. Generate each bit of the simhash as:

   (a) **if** $v[i] > 0$ **then** $simhash[i] = 1$

   (b) **else** $simhash[i] = 0$

It should be noted that a single similarity measure presents only a part of the complete picture. Two graphs may be different from each other in more than one way. Hence it may be necessary to compute multiple metrics between them to understand the complete nature of the similarity or difference.

Also, this is not an exhaustive list of similarity metrics that could be implemented for comparing semantic web graphs. Sets could be constructed from entities used in both the graphs being compared; and these sets can be used to compute a Jaccard similarity metric. Such a metric would indicate an overlap between the entities used in both the Semantic Web documents.

3.1.5 Pairwise Comparison

Given an input of semantic web documents, we need to find all pairs that are similar to each other. The total number of metrics computed for each pair of SWDs is 17: two kinds of cosine similarity (as already mentioned) and three other metrics for each reduced form pair. We adopt the following two-stage procedure to compare a pair of semantic web documents:

1. Compute the two cosine similarity values between the canonical n-triples representations (as shown in Table 3.7) of both the SWDs.

2. If the cosine similarity values are below a 0.7, then eliminate this pair from further consideration, else add this pair to a list of candidate pairs. The threshold of 0.7 was empirically determined from pilot experiments and verified for our experimental dataset. This threshold is configurable, and can be changed if required.

3. For all candidate pairs, compute the remaining three pairwise similarity metrics for each reduced form.

Thus the cosine similarity metric is used as an intial filter to reduce the remaining computations.

3.1.6 Classification

We use three different classifiers to detect various kinds of similarity between candidate pairs. In order to train these classifiers, we need a labeled training data-set. Such

Metric	**Description**
CosineSim	Cosine similarity between document vectors of the canonicalized n-triples form of each SWD (see Table 3.7)
LocalNameCosineSim	Cosine similarity between documents vectors of the canonicalized ntriples form of each SWD. The local names of terms are used as attributes.(see 3.1.4)
CanonicalJaccard	Jaccard measure computed between the canonicalized n-triples form of each SWD
NoLiteralJaccard	Jaccard measure computed between the no-literal form of each SWD (see table 3.9)
OnlyLiteralJaccard	Jaccard measure computed between the only-literal form of each SWD (see table 3.8)
LocalNameJaccard	Jaccard measure computed between the local-name form of each SWD (see table 3.10)
LocalNameNoLiteralJaccard	Jaccard measure computed between the local-name-no-literal form of each SWD (see table 3.11)
CanonicalContainment	Containment measure computed between the canonicalized n-triples form of each SWD (see table 3.7)
NoLiteralContainment	Containment measure computed between the no-literal form of each SWD (see table 3.9)
OnlyLiteralContainment	Containment measure computed between the only-literal form of each SWD (see table 3.8)
LocalNameContainment	Containment measure computed between the local-name form of each SWD (see table 3.10)
LocalNameNoLiteralContainment	Containment measure computed between the local-name-no-literal form of each SWD (see table 3.11)
CanonicalSimhash	Hamming distance between the simhashes of the canonicalized n-triples form of each SWD (see table 3.7)
NoLiteralSimhash	Hamming distance between the simhashes of the no-literal form of each SWD (see table 3.9)
OnlyLiteralSimhash	Hamming distance between the simhashes of the only-literal form of each SWD (see table 3.8)
LocalNameSimhash	Hamming distance between the simhashes of the local-name form of each SWD (see table 3.10)
LocalNameNoLiteralSimhash	Hamming distance between the simhashes of the local-name-no-literal form of each SWD (see table 3.11)

Table 3.12. Metrics computed between a pair of SWDs

Attribute	**Value**
CosineSim	0.7677
LocalNameCosineSim	0.7677
CanonicalJaccard	0.0025
NoLiteralJaccard	0.0659
OnlyLiteralJaccard	0
LocalNameJaccard	0.0025
LocalNameNoLiteralJaccard	0.0659
CanonicalContainment	0.0397
NoLiteralContainment	0.1238
OnlyLiteralContainment	0
LocalNameContainment	0.6598
LocalNameNoLiteralContainment	0.6446
CanonicalSimhash	49
NoLiteralSimhash	16
OnlyLiteralSimhash	64
LocalNameSimhash	52
LocalNameNoLiteralSimhash	26

Table 3.13. A typical feature vector for a candidate pair

a dataset is annoted with the ground truth about whether the SWDs in a given pair in the dataset are similar to each other (refer to 1.2). Accordingly, we have three labels for each pair:

1. A binary label identifying whether the candidate SWDs in the pair differ only in the literal content and are similar in terms of classes and properties used (structural similarity)

2. A binary label identifying whether the candidate SWDs in the pair differ only in the base-URI

3. A binary label identifying whether the candidate SWDs in the pair have a versioning relationship

Pairwise similarity measures are computed for each candidate pair in the labeled dataset (shown in Table 3.12). Three different feature vectors (one for each classifier) are constructed for each candidate pair, using the appropriate attributes. The attributes used are the similarity measures that have been computed. For example, a typical feature vector (for the versioning classification) might be represented as shown in Table 3.13. These classifiers are then used to detect the various forms of similarity (structural similarity, difference in base-URI, and versioning).

3.2 Computing Delta Between Two Versions

Once it is determined that two SWDs have a versioning relationship between them, we compute the set of statements which can account for the change between successive versions of the SWD. As a result, we use the concept of additive and subtractive deltas.

Additive delta *is the set of triples that are added to the older version in order to generate the newer version.* **Subtractive delta** *is the set of triples are deleted from the older version to generate the newer version.* The additive and subtractive deltas together form a graph delta. We use the following approaches:

3.2.1 Raw Delta

This method simply computes a raw delta by doing a statement-by-statement comparison between the canonical forms of the two SWDs (see Algorithm 2). If the number of triples in the two SWDs is m and n respectively, then the total number of comparisons is $O(mn)$. During the comparison, only the local names of the entities are considered i.e. we ignore the base-URI of all entities. This is done so as to account for pairs that differ only in their base-URIs. Instead, if we did take into account, the fully qualified names for all entities, then a raw delta between two graphs that differ only in base-URI will be bigger than each of the graphs. It is possible to compute the delta using this method because the canonicalization process that we apply to each SWD serves to smoothen the disparities between statements in similar documents. It assigns unique and standard identifiers to blank nodes and deterministically orders the statements.

Another approach that could be followed is to preserve all the non-base URIs in the semantic web graphs, and then do the raw-delta comparison, instead of keeping only the local names.

Algorithm 2 COMPUTE-RAW-DELTA(K, K')

```
1: Load all triples in K and K' into memory
2: for all $triple$ in $K$ do
3:    if $triple$ not in $K'$ then
4:       Add $triple$ to $additiveDelta$
5:    end if
6: end for
7: for all $triple$ in $K'$ do
8:    if $triple$ not in $K$ then
9:       Add $triple$ to $subtractiveDelta$
10:   end if
11: end for
```

Berners-Lee et al. (Berners-lee & Connolly 2004) state that “in the general case, with no constraints on how the graphs are serialized, line-oriented deltas can be as large as the data itself, even between files representing the same graph...The ordering and the identification of bnodes are the two ways which serializations of the same graph can arbitrarily differ”. We tackle the problem of ordering and identification of blank-nodes using our implementation of the canonicalization algorithm.

3.2.2 Delta After Deductive Closure

This method first computes the deductive closures of the two graphs using Jos de Roo's n3 implementation[4] of OWL rules. Then the resultant graphs are converted to their canonicalized form using Algorithm 1, and then a raw delta is generated between them using Algorithm 3.

Algorithm 3 COMPUTE-CLOSURE-DELTA(K, K')

```
1: Load all triples in K and K' into memory
2: $K \leftarrow Closure(K)$
3: $K' \leftarrow Closure(K')$
4: for all $triple$ in $K$ do
5:   if $triple$ not in $K'$ then
6:     Add $triple$ to $additiveDelta$
7:   end if
8: end for
9: for all $triple$ in $K'$ do
10:   if $triple$ not in $K$ then
11:     Add $triple$ to $subtractiveDelta$
12:   end if
13: end for
```

3.2.3 Delta at Class Level of an Ontology

This method has been implemented only for ontology version pairs that are serialized in the form of XML, but can be generalized to RDF-data document pairs as well. Instead of comparing statements in the overall graph, it compares statements at a more granular level. This kind of delta is able to pin-point the location of the change at the level of the concepts defined within an ontology. The XML representation of an ontology groups concept definitions together within XML fragments. We exploit this fact in this approach.

[4]http://eulersharp.sourceforge.net/2003/03swap/rpo-rules.n3

This approach can be applied to any other graph representation where the statements that constitute a particular concept (class or property) definition are grouped together, and can be extracted as such. The Swoogle index contains a record of 40245 ontologies (as of June 15, 2010). 34659 of these are serialized in the form of XML. This approach is a variant of the one followed by (Klein *et al.* 2002):

1. Split the XML document at the topmost level i.e. extract all the children of the root node of the XML document. Each child node represents the intended ”definition” of a concept.

2. Each of these child nodes are then parsed into groups of n-triple statements.

3. Each group of n-triple statements represents a small graph in itself. Each graph is the definition of a specific concept or property. The graphs can be identified by the ID of the concept or property being defined.

4. Using this identifier; for each graph in the one version of the ontology, locate the corresponding graph in the other version of the ontology.

5. Convert both the graphs to their canonicalized forms.

6. These two graphs can now be compared at a statement level as done before for the raw delta (see Algorithm 2).

Such a method could be generalized to RDF-data documents by grouping statements in a n-triple document by subject of the triple. Each group of statements can then be considered as a sub-graph and compared with the corresponding sub-graph in the other version.

Algorithm 4 DETECT-CLASS-RENAMING($additiveDelta$, $subtractiveDelta$)

```
1: for all $triple$ in $additiveDelta$ do
2:     Generate character n-grams for Local-Name($triple.subject$)
3:     Generate character n-grams for Local-Name($triple.object$)
4: end for
5: for all $triple$ in $subtractiveDelta$ do
6:     Generate character n-grams for Local-Name($triple.subject$)
7:     Generate character n-grams for Local-Name($triple.object$)
8: end for
9: for all $triple1$ in $additiveDelta$ do
10:     for all $triple2$ in $subtractiveDelta$ do
11:         $overlap \leftarrow Compute\text{-}Jaccard(ngrams(triple1.subject),$
            $ngrams(triple2.subject))$
12:         if $overlap > threshold$ then
13:             Add tuple $(triple1.subject, triple2.subject)$ to $subjectCandidates$
14:         end if
15:     end for
16: end for
17: for all $triple1$ in $additiveDelta$ do
18:     for all $triple2$ in $subtractiveDelta$ do
19:         $overlap \leftarrow Compute\text{-}Jaccard(ngrams(triple1.object),$
            $ngrams(triple2.object))$
20:         if $overlap > threshold$ then
21:             Add tuple $(triple1.object, triple2.object)$ to $objectCandidates$
22:         end if
23:     end for
24: end for
25: for all $(t1, t2)$ in $subjectCandidates$ do
26:     for all $triple$ in $CreateCopy(subtractiveDelta)$ do
27:         if $triple.subject \equiv t1$ then
28:             Replace $triple.subject$ by $t2$
29:             if $triple$ not in $additiveDelta$ then
30:                 Remove $(t1, t2)$ from subjectCandidates
31:             end if
32:         end if
33:     end for
34: end for
35: Perform Steps 25-34 for $objectCandidates$
36: return $subjectCandidates$ and $objectCandidates$
```

3.2.4 Heuristic Method to Detect Class Renaming

Sometimes several statements are generated in a delta as a result of a relatively simple conceptual change like the renaming of a class. This is done as shown in Algorithm 4. Essentially this algorithm computes the n-gram overlap between subjects in the $additiveDelta$ and $subtractiveDelta$ using a Jaccard coefficient calculation. For pairs of subjects where the Jaccard coefficient value is high (empirically predetermined), the pair of subjects is added to a list of candidates. A similar computation is done for all objects in the $additiveDelta$ and $subtractiveDelta$. Next, for each candidate pair of subject-class-names, all the occurrences of the old class-name in statements in the subtractiveDelta are replaced by the new class-name. Then the presence of these statements is checked in the additiveDelta. If all of the statements are actually present, then it is confirmed that the candidate tuple $(t1, t2)$ is actually an instance of a class renaming. Similarly we check for renaming of object-classes.

Chapter 4

IMPLEMENTATION AND EVALUATION

4.1 Implementation

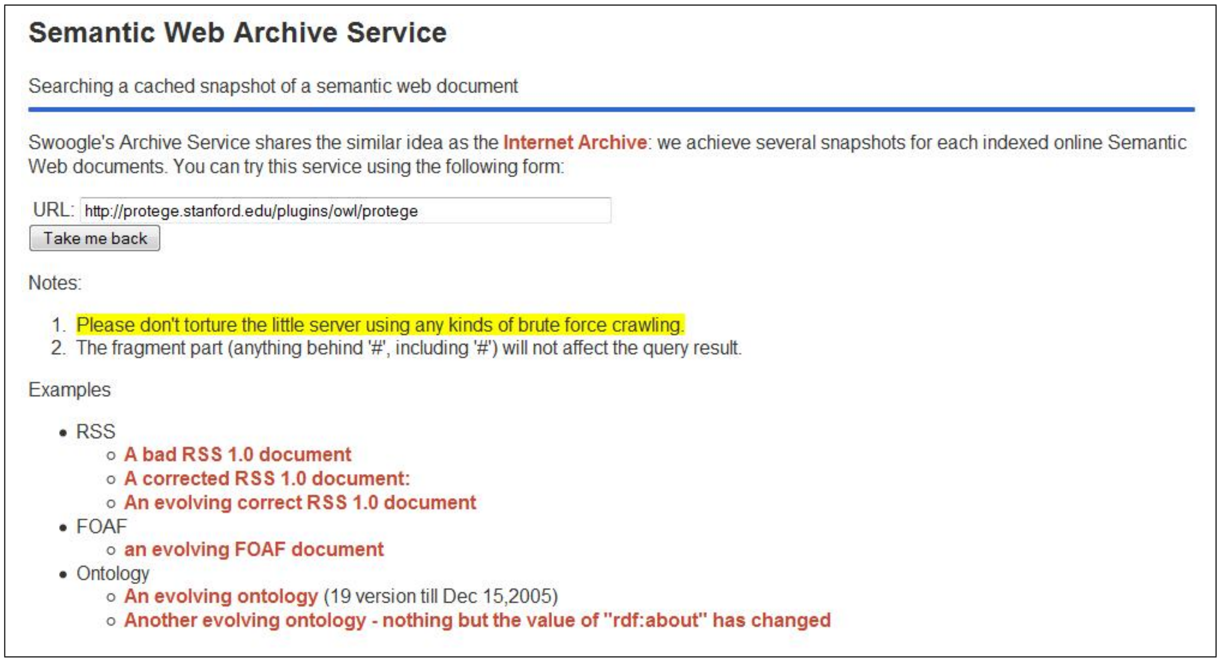
Our system *simile* was implemented using SemWeb.NET which is a Semantic Web/RDF library written in C# for Mono or Microsoft's .NET framework by Joshua Tauberer[1]. We used the MySQL relational database management system to store all our data. The system starts with a collection of semantic web documents. It first generates the canonical form, and other reduced forms for each SWD. Then it computes pairwise cosine similarity metrics for each pair. Pairs are filtered based on the values of the cosine similarity metrics. The system then computes the other similarity metrics between the candidate pairs. These metrics serve as features for machine learning based classifiers that characterize the relationship between the SWDs in each pair. If a versioning relationship is detected between the two SWDs in a pair, then a delta is generated between them.

4.2 Construction of Data-set

Our system is based on particular kinds of similarity that have been observed manually from Swoogle's repository, and these are not formally specified. In addition, there

[1]http://razor.occams.info/code/semweb/

exists no standard labeled dataset of similar semantic web documents that we could use for the purpose of evaluating our system. Hence we decided to create a collection of semantic web documents from Swoogle's semantic web archive and cache services[2]. Swoogle periodically crawls the semantic web and maintains several snapshots for each indexed SWD.


Semantic Web Archive Service

Searching a cached snapshot of a semantic web document

Swoogle's Archive Service shares the similar idea as the **Internet Archive**: we achieve several snapshots for each indexed online Semantic Web documents. You can try this service using the following form:

URL: http://protege.stanford.edu/plugins/owl/protege

Notes:

1. Please don't torture the little server using any kinds of brute force crawling.
2. The fragment part (anything behind '#', including '#') will not affect the query result.

Examples

- RSS
  - **A bad RSS 1.0 document**
  - **A corrected RSS 1.0 document:**
  - **An evolving correct RSS 1.0 document**
- FOAF
  - **an evolving FOAF document**
- Ontology
  - **An evolving ontology** (19 version till Dec 15,2005)
  - **Another evolving ontology - nothing but the value of "rdf:about" has changed**

FIG. 4.1. Swoogle Semantic Web Archive

Swoogle's Semantic Web Archive Service shows the evolution of a semantic web document through multiple versions. Swoogle assigns a unique ID to each URL from which a semantic web document is crawled. All the versions extracted from this URL are given the same ID. Multiple versions of a SWD are differentiated by the date on which they were discovered. We look through the Swoogle archive to find SWDs whose multiple versions have been saved in the cache. We add such versions to our data-set and label them

[2]http://swoogle.umbc.edu/index.php?option=com_swoogle_service&service=archive

as having a versioning relationship.

4.3 Computation of Similarity Metrics

We compute pairwise similarity metrics for each pair of semantic web documents in our input collection. This is approximately $O(n^2)$ comparisons where n is the total number of input semantic web documents in the collection. This is done as described in section 3.1.5. The cosine similarity measures are used as an initial filter before computing the rest of the metrics. Eventually, we are left with 17 metrics for each candidate pair as shown in Table 3.12

4.4 Detecting Similarity Between Pairs

We computed similarity metrics between candidate pairs of SWDs in the corpus. These metrics are then used to characterize the type of relation between the two SWDs in a candidate pair:

1. Same classesa and properties used, but differ only in literal content
2. Differ only in base-URIs used
3. Have a versioning relationship between them

4.4.1 Detecting Pairs that Differ only in Literal Content

Table 1.1 showed an example of two SWDs with the same structure (i.e. use the same kind of classes and properties) but different literal content. This is one of the types of similarity that our system detects. For detecting this kind of structural similarity between a pair of SWDs, we build a feature vector using the following measures (refer to Table for 3.12 details):

Result Metric	**Value**
Correct Classified Instances	1590 (98.6532%)
Incorrectly Classified Instances	22 (1.3648%)
Kappa statistic	0.9727
Mean absolute error	0.0136
Root mean squared error	0.1149
Relative absolute error	2.7162 %
Root relative squared error	22.9795 %
Total Number of Instances	1612

Table 4.1. Stratified cross-validation (Naive Bayes) summary for similarity in classes and properties used

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.973	0	1	0.973	0.986	1	yes
	1	0.027	0.973	1	0.987	0.996	no
Weighted Avg.	0.986	0.014	0.987	0.986	0.986	0.998	

Table 4.2. Detailed Accuracy by class (structural similarity), using Naive Bayes

1. CosineSim
2. LocalNameNoLiteralJaccard
3. LocalNameNoLiteralSimhash

Manual Attribute Selection We chose these measures because they are the most relevant for measuring similarity in classes and properties used. For performing this experiment, we used a set of 402 semantic web documents (over 161,000 candidate pairs) downloaded from the Swoogle archive. We identified 806 pairs where the two candidates used the same classes and properties, but different only in the literal data. We used FOAF profiles that were machine generated, as well as snapshots of chat transcripts in RDF format, and snapshots of other files generated by content management systems. We labeled such pairs as positive. Such automatically generated RDF documents have a well defined structure in terms of classes and properties used. We also identified an equal number of negative pairs. Next, we construct feature vectors from the measures mentioned above. Use built a Naive-Bayes classifier on the 1612 feature vectors (50% positive, 50% negative). A ten-fold stratified cross-validation on this data-set yielded the results as shown in Tables 4.1 and 4.2

Including the feature LocalNameCosineSim slightly improves the true positive rate of the classifier (3 more correctly identified). Tables 4.3 and 4.4 show the 10-fold stratified cross-validation results.

Using All Attributes We performed another experiment where we used all of the 17 features to let the SVM machine-learning algorithm decide which of the features are most relevant, rather than manually picking the features ourselves. The results of a ten-fold stratified cross-validation using a SVM on this dataset are as shown in Tables 4.5, and 4.6.

An attribute relevance ranking performed using the Infogain Attribute Evaluator and the Ranker search method in Weka ranked the features as shown in Table 4.7. (showing the top six results). The results of the attribute ranking confirmed our initial intuition about the manually selected attributes.

Figures 4.2, 4.3 , 4.4, and 4.5 illustrate how the positive and negative class labels are distributed by some of the attributes of the feature vector for this classification. From the figures we can clearly see that most of these attributes have the ability to divide the data-set into positive and negative examples. This explains the high precision and recall of the classifier.

Result Metric	**Value**
Correct Classified Instances	1593 (98.8213 %)
Incorrectly Classified Instances	19 (1.1787 %)
Kappa statistic	0.9764
Mean absolute error	0.0118
Root mean squared error	0.1049
Relative absolute error	2.3618 %
Root relative squared error	20.9806 %
Total Number of Instances	1612

Table 4.3. Stratified cross-validation (Naive Bayes) summary for similarity in classes and properties used (including LocalNameCosineSim)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.976	0	1	0.976	0.988	1	yes
	1	0.024	0.977	1	0.988	0.994	no
Weighted Avg.	0.988	0.012	0.988	0.988	0.988	0.997	

Table 4.4. Detailed Accuracy by class (structural similarity - including feature LocalNameCosineSim) using Naive Bayes

Result Metric	**Value**
Correct Classified Instances	1598 (99.1315%)
Incorrectly Classified Instances	14 (0.8685%)
Kappa statistic	0.9826
Mean absolute error	0.0087
Root mean squared error	0.0932
Relative absolute error	1.737 %
Root relative squared error	18.6385 %
Total Number of Instances	1612

Table 4.5. Stratified cross-validation (SVM-linear-kernel) summary for similarity in classes and properties used

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.995	0.012	0.988	0.995	0.991	0.991	yes
	0.988	0.005	0.995	0.988	0.991	0.991	no
Weighted Avg.	0.991	0.009	0.991	0.991	0.991	0.991	

Table 4.6. Detailed Accuracy by class (structural similarity), using SVM-linear kernel

0.98	NoLiteralSimhash
0.9799	LocalNameNoLiteralSimhash
0.9555	LocalNameCosineSim
0.948	CosineSim
0.9262	LocalNameNoLiteralJaccard
0.9261	NoLiteralJaccard

Table 4.7. Attribute relevance ranking using Infogain Attribute Evaluator and Ranker search for similarity in classes and properties used (top-six)

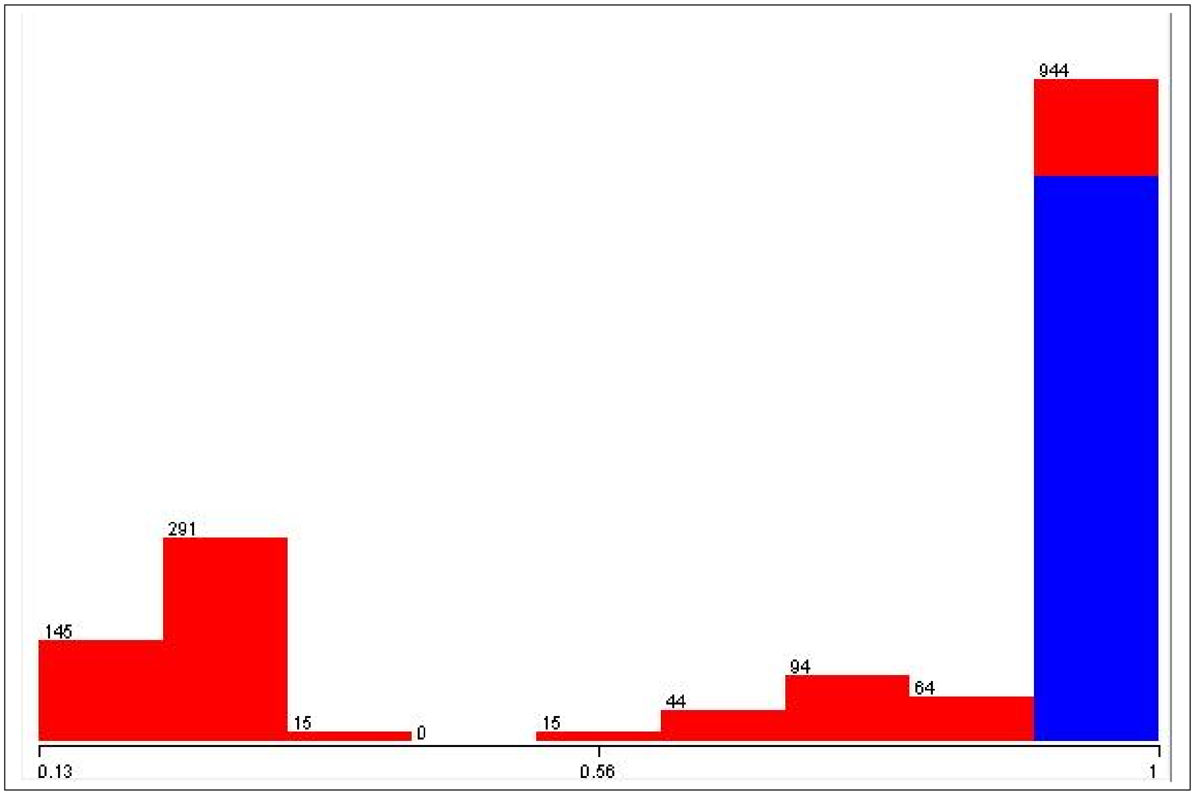


FIG. 4.2. Effect of CosineSim on structural similarity. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances.

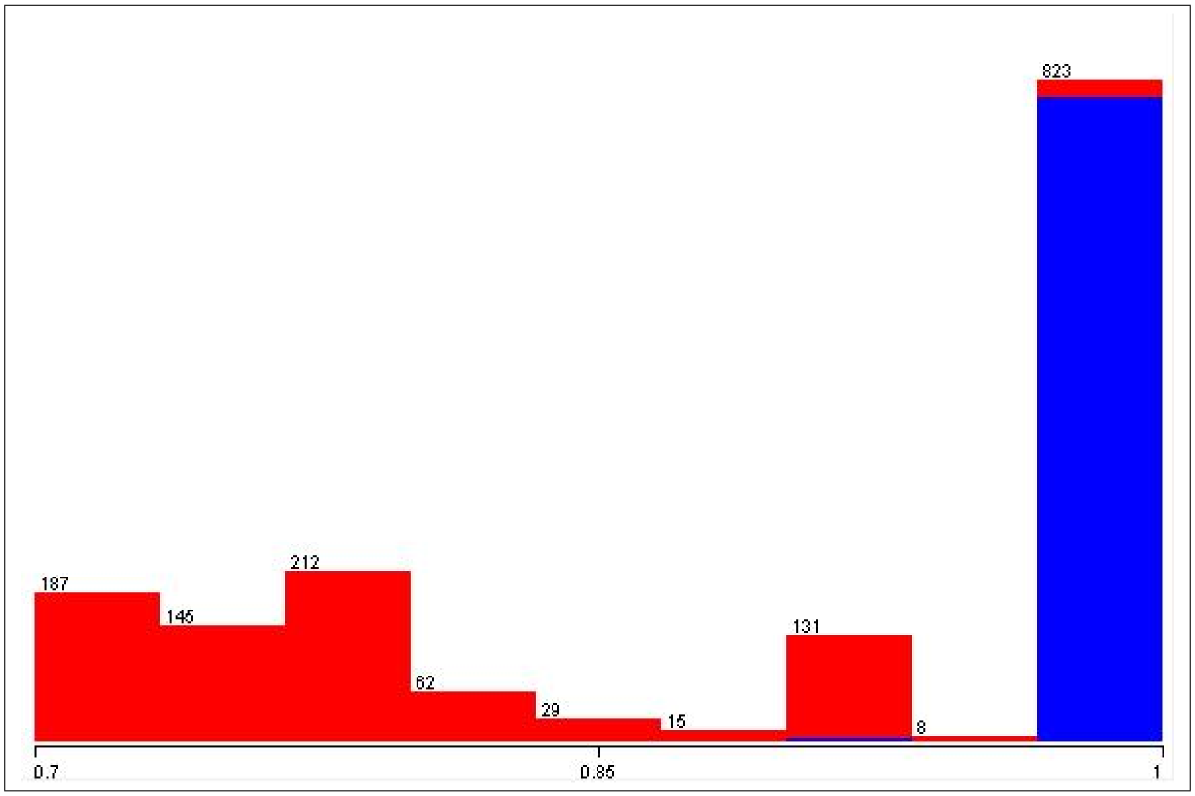


FIG. 4.3. Effect of LocalNameCosineSim on structural similarity. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances.

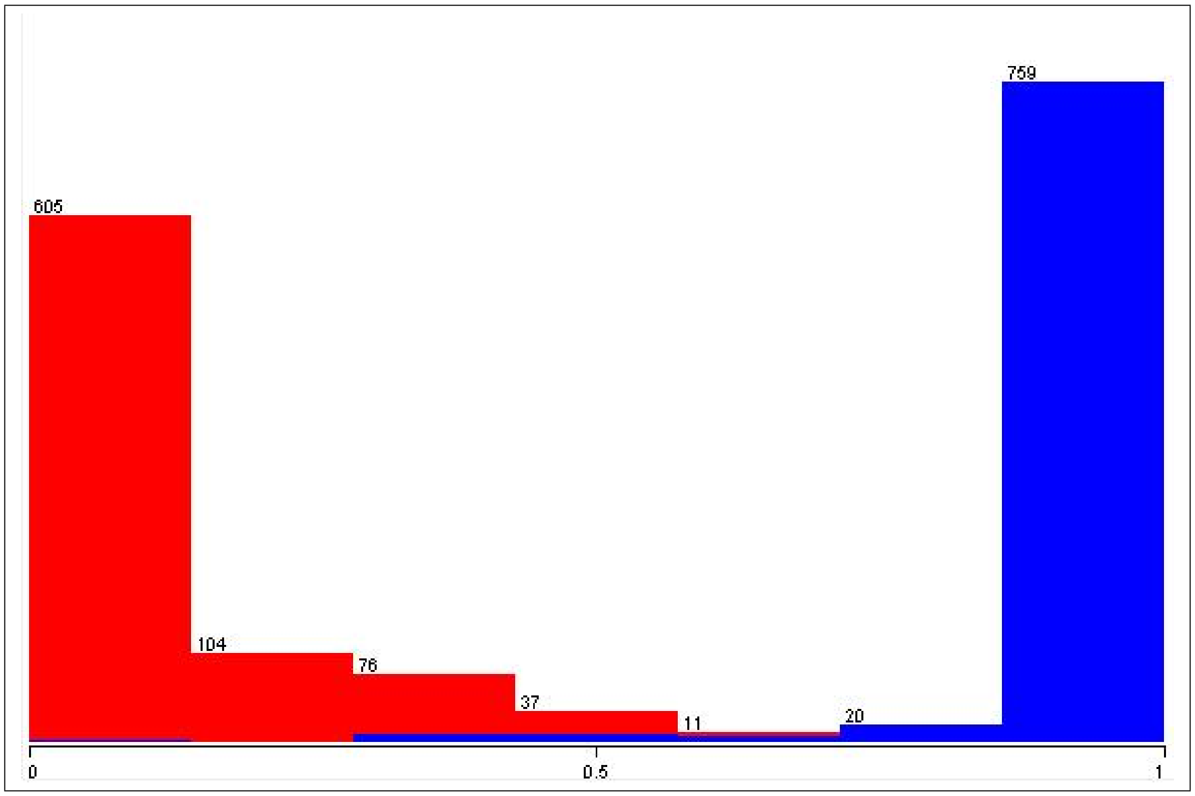


FIG. 4.4. Effect of LocalNameNoLiteralJaccard on structural similarity. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances.

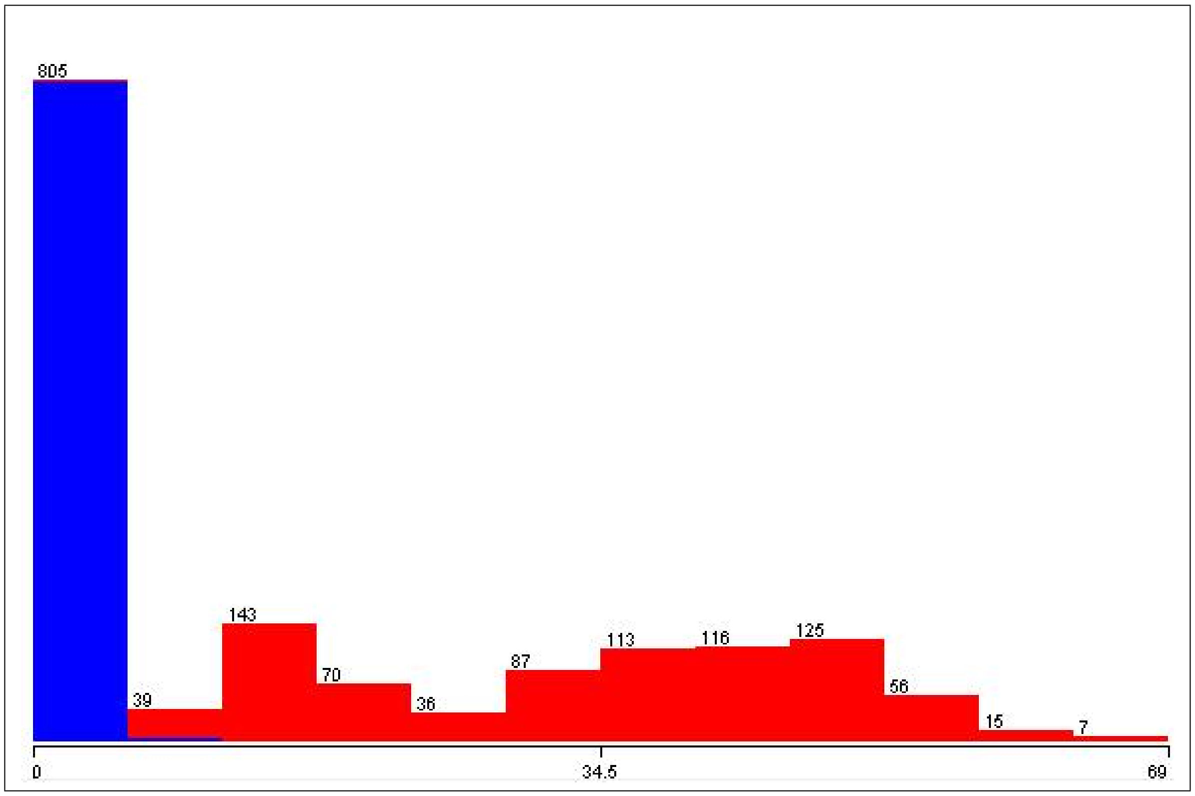


FIG. 4.5. Effect of LocalNameNoLiteralSimhash on structural similartiy. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances.

4.4.2 Detecting Pairs that Differ Only in Base-URI

An example of this use-case was shown in Fig 1.2. In order to detect similarity of this kind between a candidate pair, the metrics we use are (refer to Table for 3.12 details):

1. CosineSim
2. LocalNameCosineSim
3. LocalNameNoLiteralJaccard

Result Metric	**Value**
Correct Classified Instances	196 (98%)
Incorrectly Classified Instances	4 (2%)
Kappa statistic	0.96
Mean absolute error	0.02
Root mean squared error	0.1414
Relative absolute error	4 %
Root relative squared error	28.2843 %
Total Number of Instances	200

Table 4.8. Stratified cross-validation (Naive Bayes) summary for pairs that differ only in Base URI

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	0.04	0.962	1	0.98	0.979	yes
	0.96	0	1	0.96	0.98	0.99	no
Weighted Avg.	0.98	0.02	0.981	0.98	0.98	0.985	

Table 4.9. Detailed Accuracy by class (pairs different only in base URI), using Naive Bayes

4. LocalNameNoLiteralContainment

5. OnlyLiteralContainment

6. OnlyLiteralJaccard

These measures compare the local names of the classes and properties used, and the textual (literal) content. For performing this experiment, we used the same set of SWDs as 4.4.1. We picked 100 random SWDs, and created pairs from each SWD as follows:

1. Create a copy of the SWD, say *c*

2. Determine the most frequently occurring base-URI in the SWD, say *b1*

3. Replace all occurrences of *b1* in *c*

4. Label the original SWD and *c* as similar

Result Metric	**Value**
Correct Classified Instances	194 (97%)
Incorrectly Classified Instances	6 (3%)
Kappa statistic	0.94
Mean absolute error	0.03
Root mean squared error	0.1732
Relative absolute error	6 %
Root relative squared error	34.361 %
Total Number of Instances	200

Table 4.10. Stratified cross-validation summary (SVM-linear kernel) for pairs that differ only in Base URI

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	0.06	0.943	1	0.971	0.97	yes
	0.94	0	1	0.94	0.969	0.97	no
Weighted Avg.	0.97	0.03	0.972	0.97	0.97	0.97	

Table 4.11. Detailed Accuracy by class (pairs different only in base URI), using SVM - linear kernel

Thus, after this process, we generated 100 positive pairs, and combined them with a 100 negative pairs to generate a data-set of 200 examples. We then constructed feature vectors using the features listed above. We built a Naive-Bayes classifier on the 200 feature vectors. A ten-fold stratified cross-validation on this data-set yielded the results shown in Tables 4.8 and 4.9. A Support Vector Machine (linear kernel) trained using the same set of feature vectors did slightly worse than the Naive-Bayes classifier. The results are as shown in Tables 4.10 and 4.11.

We tried this classifier on a set of ontologies comprising the wine[3], baseball[4], geospecies[5] , and dbpedia[6] ontologies to ensure that this classifier can identify cases that occur in the real world. The classifier successfully identified each pair. Figures 4.6, 4.7,

[3]http://w3.org/2001/sw/WebOnt/guide-src/wine
[4]http://www.damn.org/2001/08/baseball/baseball-ont
[5]http://rdf.geospecies.org/ont/gsontology
[6]http://downloads.dbpedia.org/3.2/en/dbpedia-ontology.owl

and 4.8 illustrate how some of the attributes selected for classification divide the dataset into positive and negative instances. This explains the high precision and recall of the classifier results.

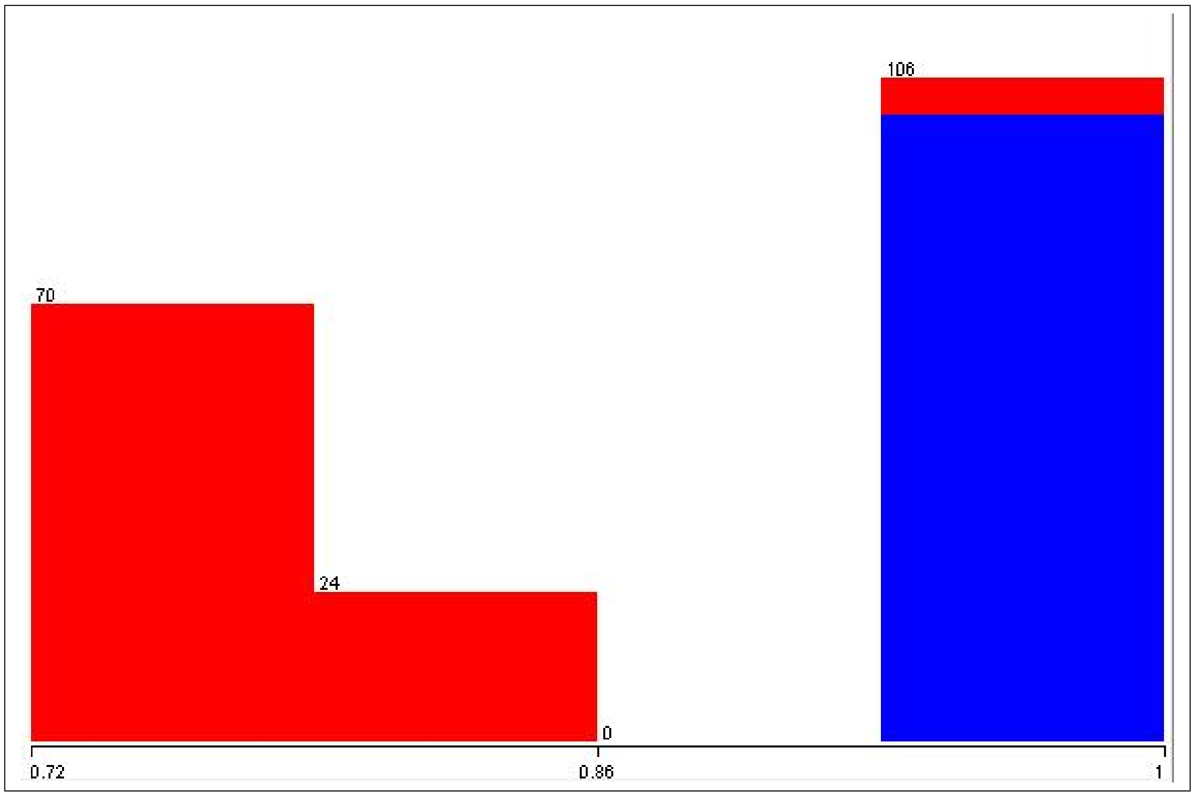


FIG. 4.6. Effect of LocalNameCosineSim on SWDs that differ only in base-URI. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances.

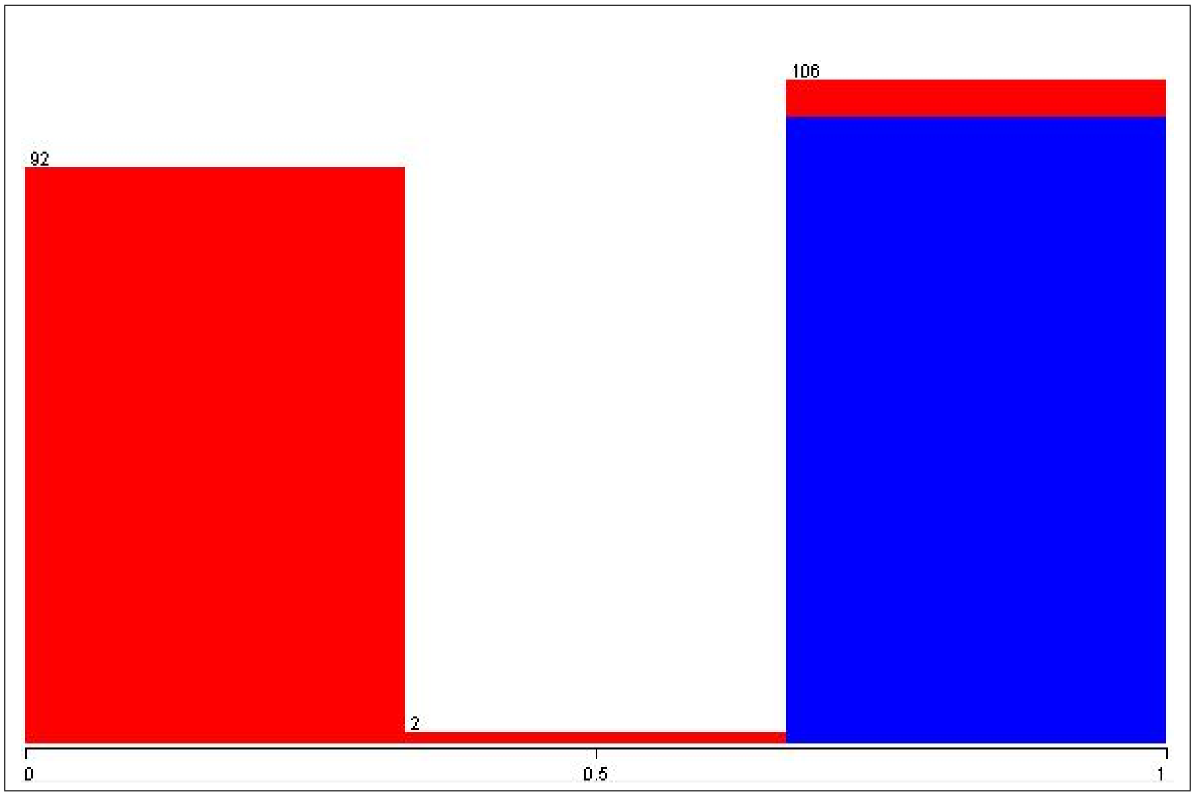


FIG. 4.7. Effect of LocalNameNoLiteralContainment on SWDs that differ only in base-URI. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances.

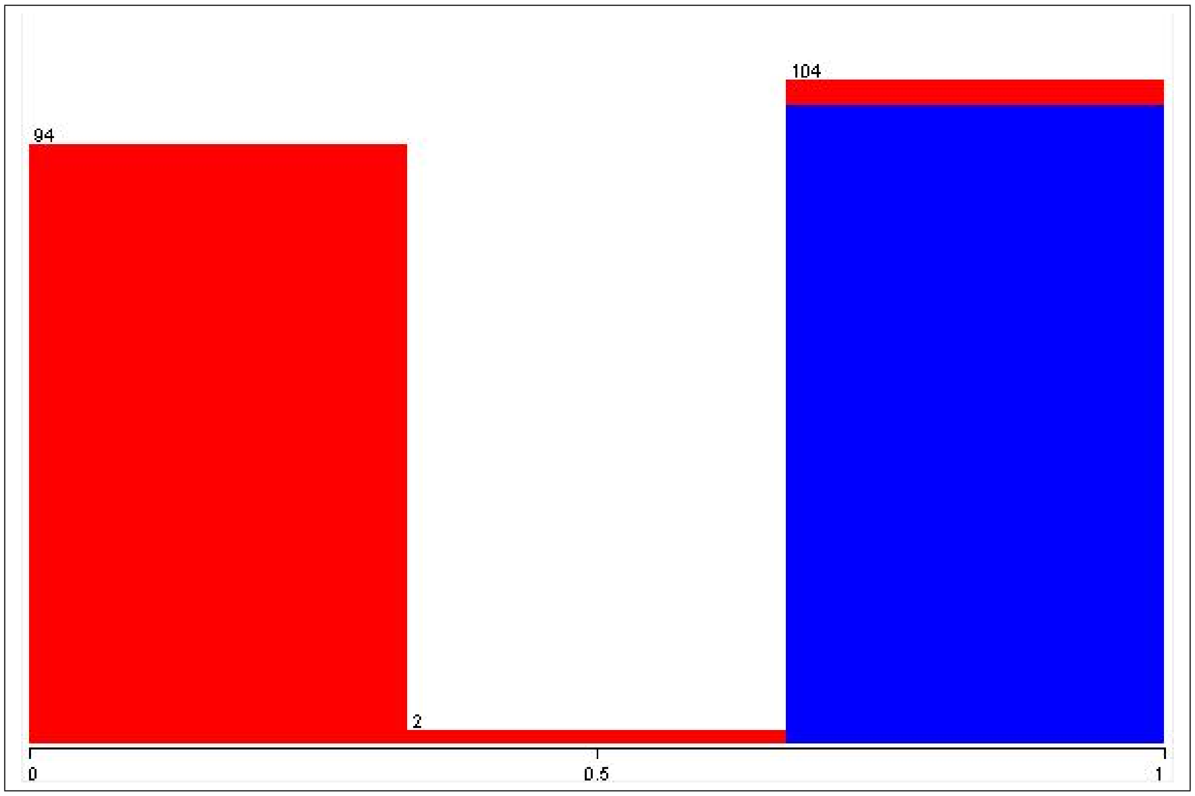


FIG. 4.8. Effect of OnlyLiteralJaccard on SWDs that differ only in base-URI. Blue(dark gray) color indicates positive label, while red(light gray) indicates negative label. Horizontal axis indicates the feature values and vertical axis indicates the number of instances.

4.4.3 Detecting Pairs with a Versioning Relationship

As mentioned already in Section 4.2, we used the Swoogle cache to get snapshots of semantic web documents at different instances of time. The time difference between two snapshots is not uniform for all the URLs. Hence some pairs have a long time-difference between then whereas some others have a shorter one. From an analysis of the dataset, we discovered what has been said earlier about semantic web documents on the web, and web documents in general (Heflin 2001): *Some pages are fairly static, others change on* *a regular basis and still others change at unpredictable intervals. These changes may vary in significance: although the addition of punctuation, correction of spelling errors, or reordering of a paragraph does not affect the semantic content of a document; other changes may completely alter meaning, or even remove large amounts of information.*

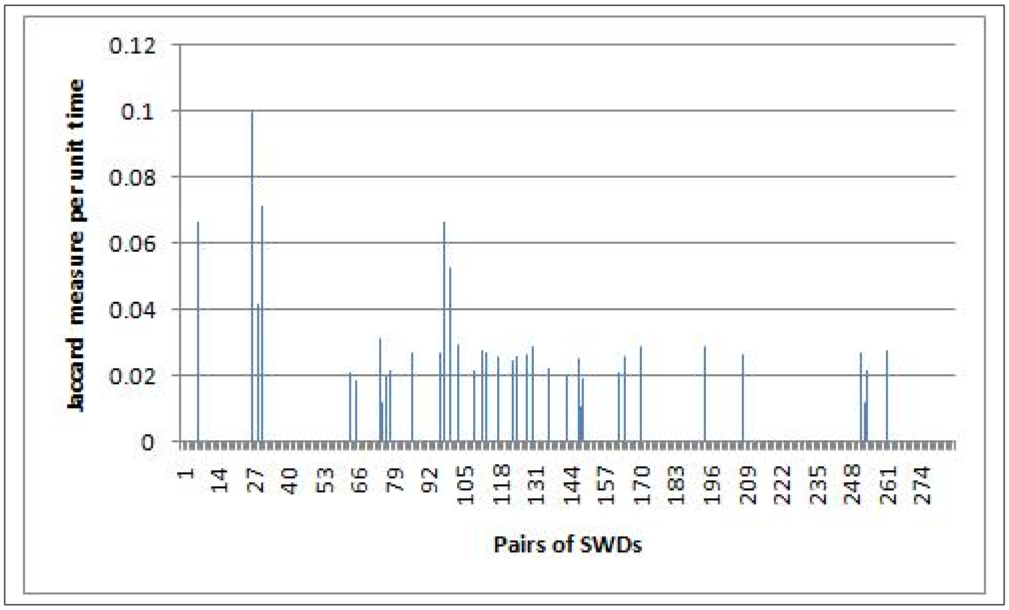


FIG. 4.9. Each point on the horizontal axis represents a pair of SWDs which are snapshots of the same URL

To measure the amount of change between two snapshots of the same semantic web document, we computed the Jaccard measure between the two snapshots and normalized it by the time interval between them. A histogram plot of this information is shown in Figure 4.9. The sparse histogram indicates that most SWDs in the collection are highly dynamic and have a huge difference across two snapshots. Our sytem cannot determine a versioning relationship between these snapshots. But the other pairs that have a relatively smaller amount of change between them can actually be considered as having a versioning

Result Metric	**Value**
Correct Classified Instances	160 (90.9091%)
Incorrectly Classified Instances	16 (9.0909%)
Kappa statistic	0.8182
Mean absolute error	0.0909
Root mean squared error	0.3015
Relative absolute error	18.1818 %
Root relative squared error	60.3023 %
Total Number of Instances	176

Table 4.12. Evaluation on test set - summary (SVM-linear kernel) for pairs with a versioning relationship

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.864	0.045	0.95	0.864	0.905	0.909	yes
	0.955	0.136	0.875	0.955	0.913	0.909	no
Weighted Avg.	0.909	0.091	0.913	0.909	0.909	0.909	

Table 4.13. Detailed Accuracy by class (versioning relationship), using SVM - linear kernel

relationship. Accordingly, we filtered the highly dynamic pairs from our dataset, and used the remaining pairs to train a Support Vector Machine (linear kernel). We used all 17 attributes to build the feature vectors for this purpose. The number of training instances was 124 (50% positive, and 50% negative). We used this SVM to classify instances from a different test data-set that was built the same way as the training data-set. The results of the classification are as shown in Tables 4.12 and 4.13.

4.5 Correctness of Delta Computation

Zeginis et al. (Zeginis, Tzitzikas, & Christophides 2007) define the following notion of correctness for RDF deltas: If Δ_x is a comparison function and Π_y be a change operation semantics, then it holds that: A pair (Δ_x, Π_y) is *correct* if, for any pair of knowledge bases K and K' it holds that $\Delta_x(\text{K}\rightarrow\text{K'})^{\Pi_y} K \Leftrightarrow K'$

We test for correctness of the deltas that we generate by verifying this condition. We apply the generated delta to the first SWD and compare the resultant, statement by statement to the second SWD. Please refer to the appendix (section A.1) for samples of the deltas generated.

Chapter 5

CONCLUSION AND FUTURE WORK

In this thesis, we explored some of the ways in which semantic web graphs may be similar to each other. We developed a system *simile* that can recognize when two semantic web graphs are similar and characterizes their difference in three ways:

1. Whether the two graphs use the same classes and properties; and differ only in the literal content used

2. Whether the two graphs differ only in the base-URI used

3. Whether the two graphs are versions i.e. one has evolved from the other

As a part of this, we implement a canonicalization algorithm for semantic web graphs that can deterministically order the statements in a graph and provide consistent identifiers to blank nodes. We generate reduced forms for each semantic web graph which are used to compute text based similarity metrics between candidate pairs. These similarity metrics are then used to characterize the candidate pair as mentioned above. Further, for pairs that have a versioning-relationship between them, our system generates a delta in terms of triples to be added or deleted (i.e. additive delta and subtractive delta). For ontology pairs, in addition to generating a raw delta, the system also generates a delta at the class/property level. Additionally, we use an n-gram overlap based heuristic method to detect situations where the statements in the delta between two ontologies can be accounted for by a relatively simple conceptual change like the renaming of a class.

5.1 Future Directions

5.1.1 Scalability

Currently our system can deal with only a small data-set. This is because we perform $O(n^2)$ comparisons (at-least for initial filtering) between all the pairs of graphs in the data-set. It is an algorithmic challenge to perform this similarity join in an efficient and scalable way. We wish to extend our system so that it can detect pairs of similar SWDs in a large corpus like that of Swoogle. (Xiao *et al.* 2008) present a prefix-filtering based approach to deal with this problem. We seek to implement a filtering mechanism like this in our system so that we can increase scalability.

5.1.2 Content of Delta Generated

We generate a statement-by-statement delta at the graph level and at the concept/class level. In each of the cases, we do not guarantee a small-sized delta. Ideally, we would like to generate the smallest possible set of triples to be added or deleted that can account for the difference between the two graphs. Sometimes a small change at a conceptual level may trigger a large delta at the triple level. We deal with one such example: when a class is renamed. We would like to develop a more deterministic method that would guarantee a small delta that could explain the change between two versions. It would also be a useful thing to know which changes can be mapped to the A-Box, and which changes can be mapped to the T-Box of the semantic web graph.

5.1.3 Standard Ontology to Describe the Relationship Between Two SWDs

Currently we group the statements in the delta as additive and subtractive deltas. Additionally, we don't have a standard way for representing information such as a class renaming (refer 3.2.4). The Graph Update Ontology[1] is a lightweight ontology to describe RDF graph updates on a statement-level. However, the GUO does not define classes or properties that can describe conceptual changes like class renaming etc. We would like to use a standard ontology that can describe the delta between two semantic web graphs, including concept level changes.

Additionally, we would like to use a standard ontology to provide a complete description of the comparison between two SWDs in a pair. Such an ontology would define concepts for all the kinds of information that we generate between a pair i.e. information about the three kinds of similarity that we detect, and the delta (if applicable).

5.1.4 Predicting Direction of Change Between Two Versions

We are already able to identify pairs of SWDs from a collection that have a versioning relationship between them. A natural addition to this feature would be identifying the older and newer versions within the pair. This could be done done by training a classifier with adequate number of training examples. The features used could be the same as the features used for detecting the versioning relationship.

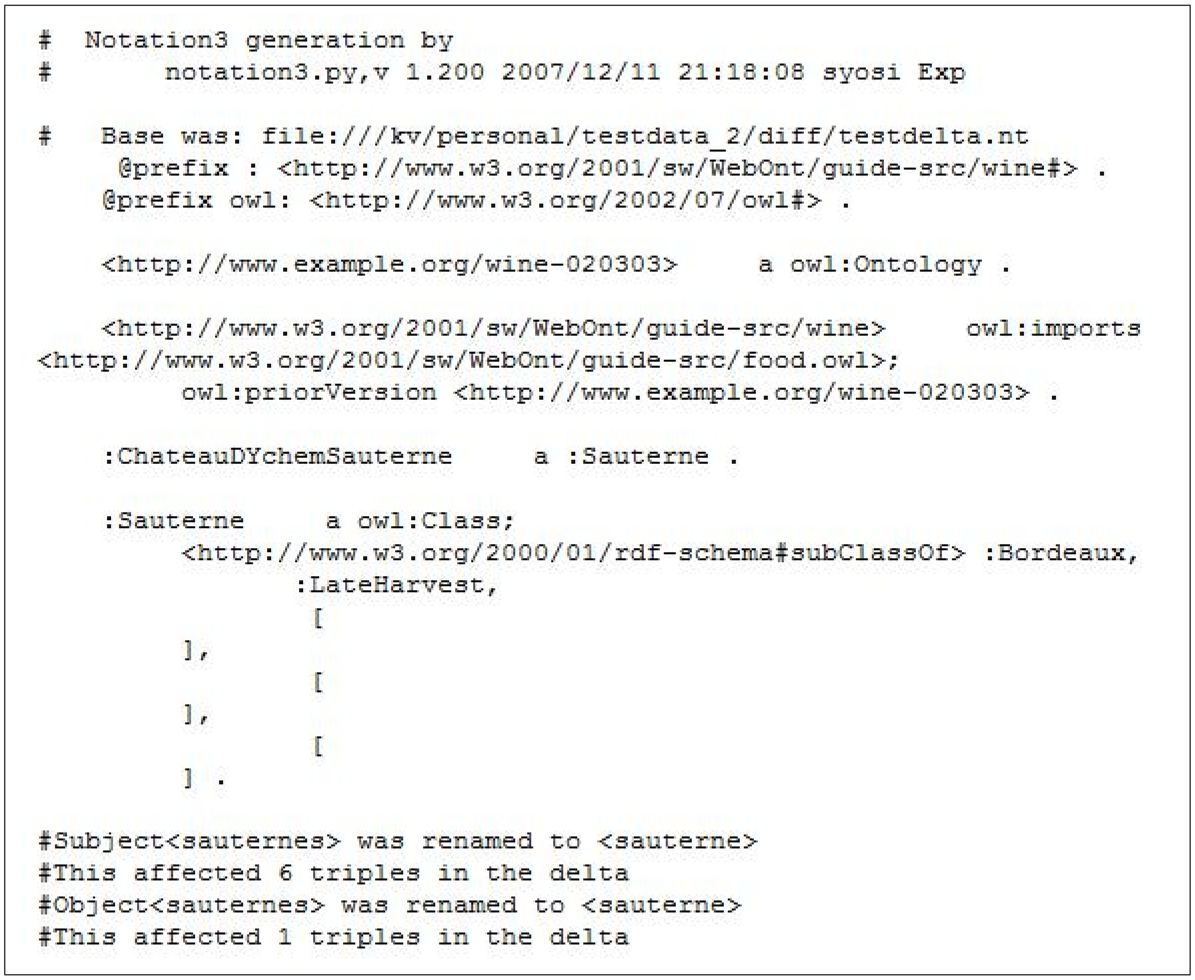
[1]http://webr3.org/specs/guo/

Appendix A

APPENDIX

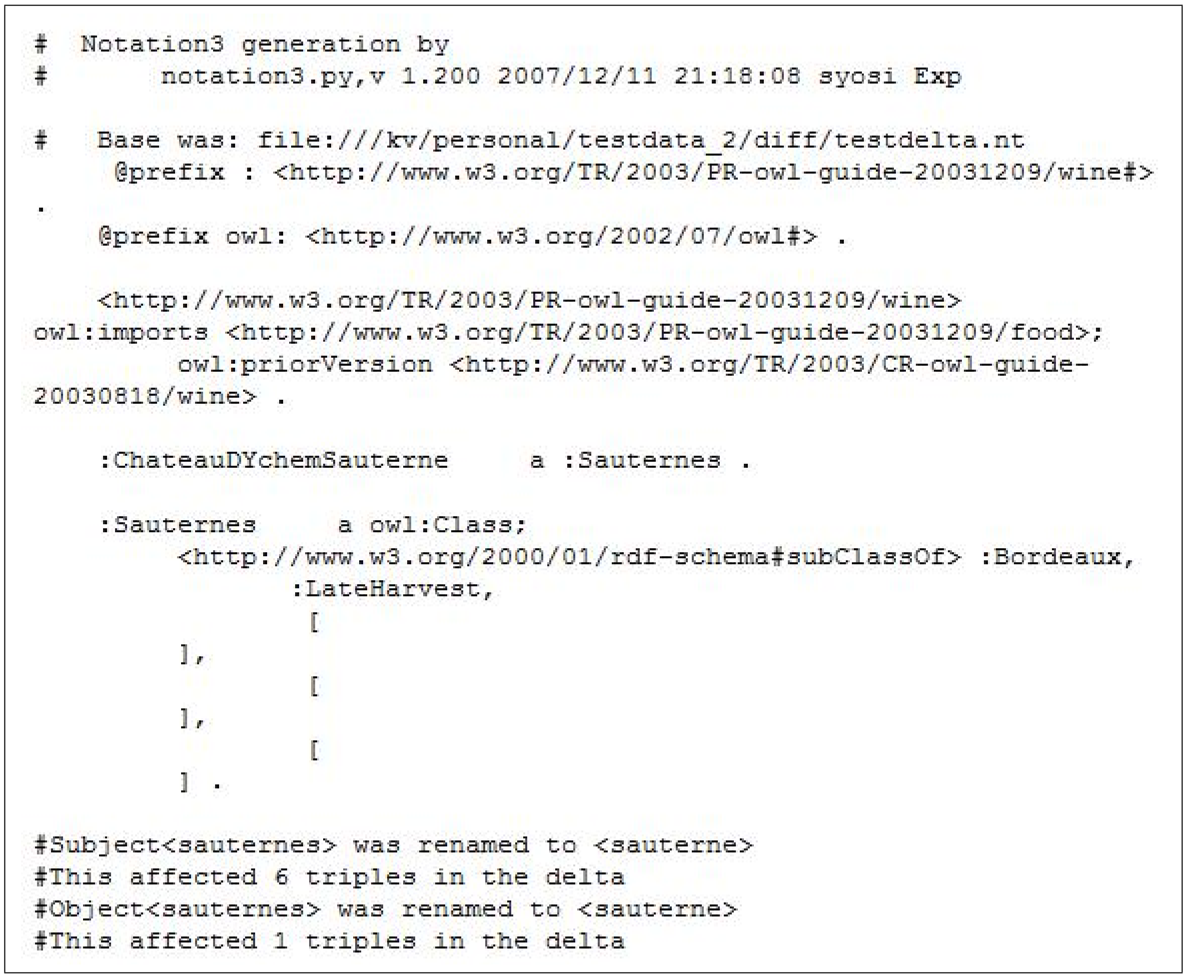
A.1 Sample Delta Outputs

The following delta were generated between two versions of the wine ontology: Figures A.1, and A.2 illustrate the raw deltas along with the class-renaming heuristic.



```
#  Notation3 generation by
#       notation3.py,v 1.200 2007/12/11 21:18:08 syosi Exp

#   Base was: file:///kv/personal/testdata_2/diff/testdelta.nt
     @prefix : <http://www.w3.org/2001/sw/WebOnt/guide-src/wine#> .
    @prefix owl: <http://www.w3.org/2002/07/owl#> .

    <http://www.example.org/wine-020303>     a owl:Ontology .

    <http://www.w3.org/2001/sw/WebOnt/guide-src/wine>     owl:imports
<http://www.w3.org/2001/sw/WebOnt/guide-src/food.owl>;
         owl:priorVersion <http://www.example.org/wine-020303> .

    :ChateauDYchemSauterne     a :Sauterne .

    :Sauterne     a owl:Class;
         <http://www.w3.org/2000/01/rdf-schema#subClassOf> :Bordeaux,
                :LateHarvest,
                 [
         ],
                 [
         ],
                 [
         ] .

#Subject<sauternes> was renamed to <sauterne>
#This affected 6 triples in the delta
#Object<sauternes> was renamed to <sauterne>
#This affected 1 triples in the delta
```

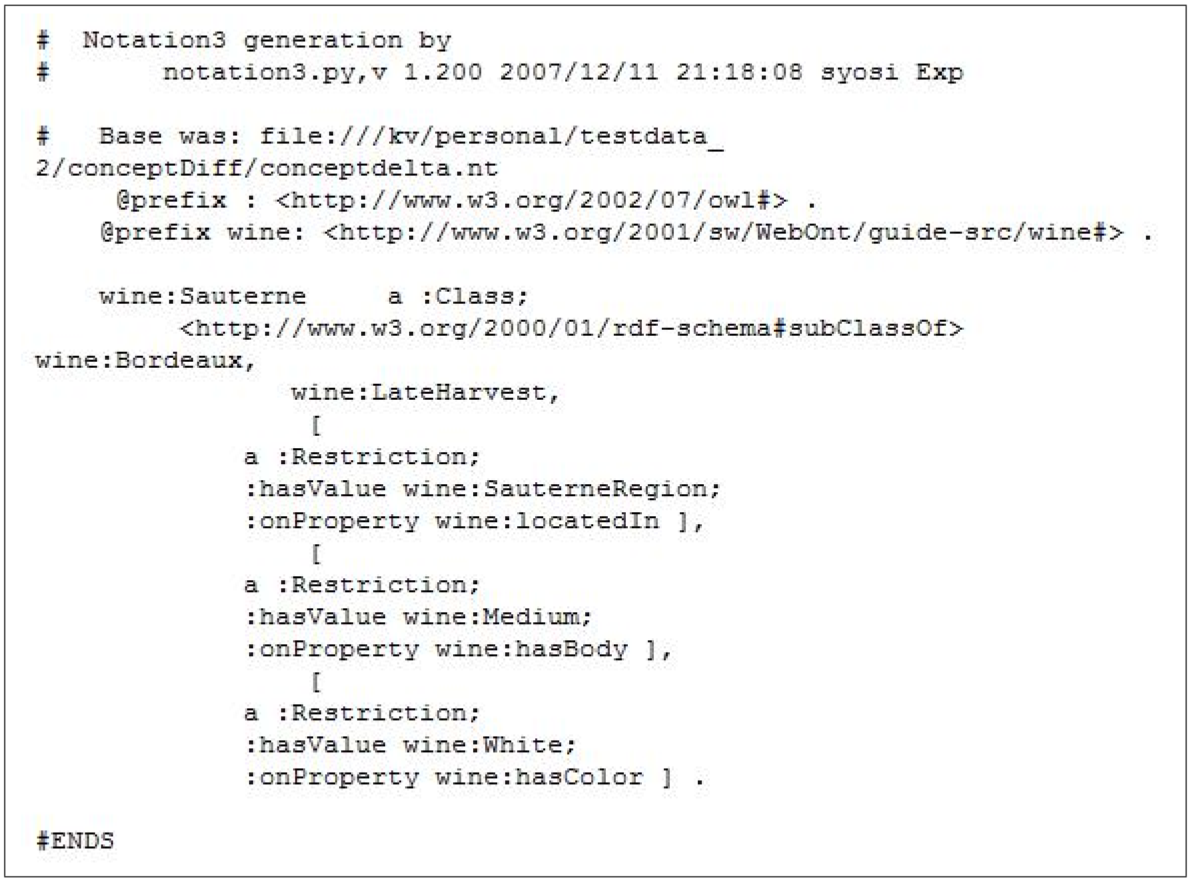
FIG. A.1. Wine raw additive delta. Note the comment about the class renaming detected



```
#  Notation3 generation by
#       notation3.py,v 1.200 2007/12/11 21:18:08 syosi Exp

#   Base was: file:///kv/personal/testdata_2/diff/testdelta.nt
     @prefix : <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
.
    @prefix owl: <http://www.w3.org/2002/07/owl#> .

    <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine>
owl:imports <http://www.w3.org/TR/2003/PR-owl-guide-20031209/food>;
         owl:priorVersion <http://www.w3.org/TR/2003/CR-owl-guide-
20030818/wine> .

    :ChateauDYchemSauterne     a :Sauternes .

    :Sauternes     a owl:Class;
         <http://www.w3.org/2000/01/rdf-schema#subClassOf> :Bordeaux,
                :LateHarvest,
                 [
         ],
                 [
         ],
                 [
         ] .

#Subject<sauternes> was renamed to <sauterne>
#This affected 6 triples in the delta
#Object<sauternes> was renamed to <sauterne>
#This affected 1 triples in the delta
```

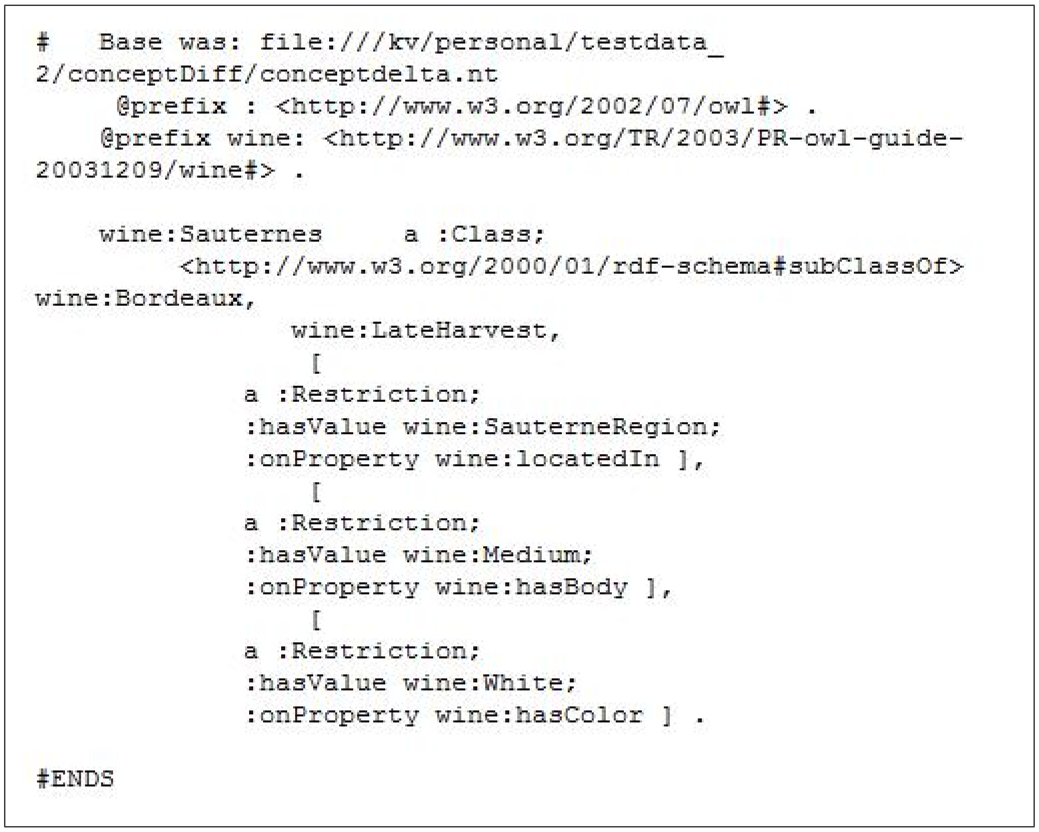
FIG. A.2. Wine raw subtractive delta. Note the comment about the class renaming detected

The figures A.3, A.4, A.6, and A.5 illustrate the concept-level deltas generated (both additive and subtractive)



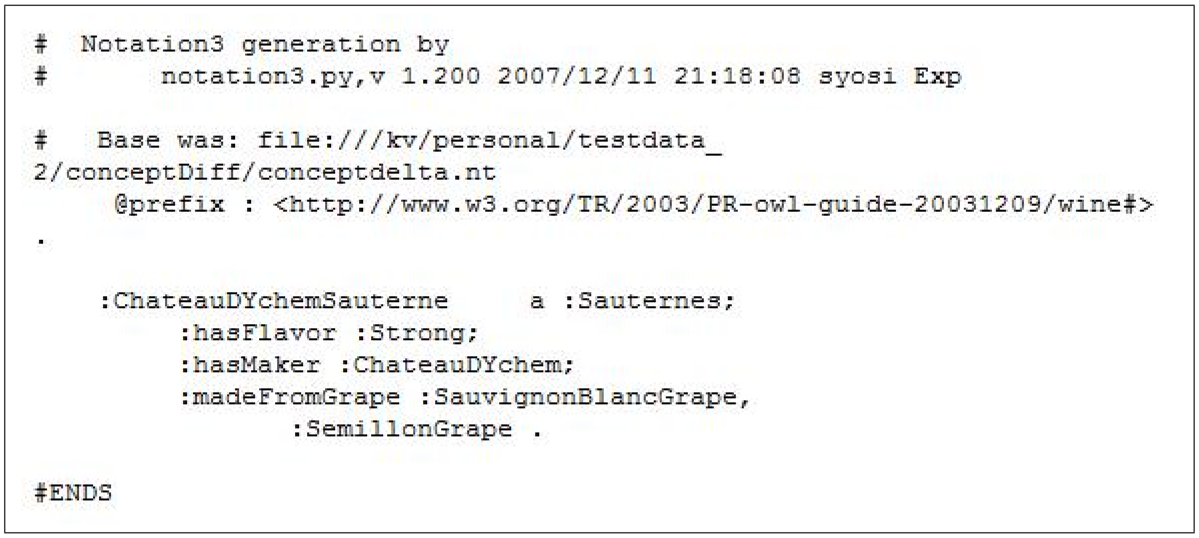
```
#  Notation3 generation by
#       notation3.py,v 1.200 2007/12/11 21:18:08 syosi Exp

#   Base was: file:///kv/personal/testdata_
2/conceptDiff/conceptdelta.nt
     @prefix : <http://www.w3.org/2002/07/owl#> .
    @prefix wine: <http://www.w3.org/2001/sw/WebOnt/guide-src/wine#> .

    wine:Sauterne     a :Class;
         <http://www.w3.org/2000/01/rdf-schema#subClassOf>
wine:Bordeaux,
                wine:LateHarvest,
                 [
             a :Restriction;
             :hasValue wine:SauterneRegion;
             :onProperty wine:locatedIn ],
                 [
             a :Restriction;
             :hasValue wine:Medium;
             :onProperty wine:hasBody ],
                 [
             a :Restriction;
             :hasValue wine:White;
             :onProperty wine:hasColor ] .

#ENDS
```

FIG. A.3. Concept level delta (sauterne-additive)



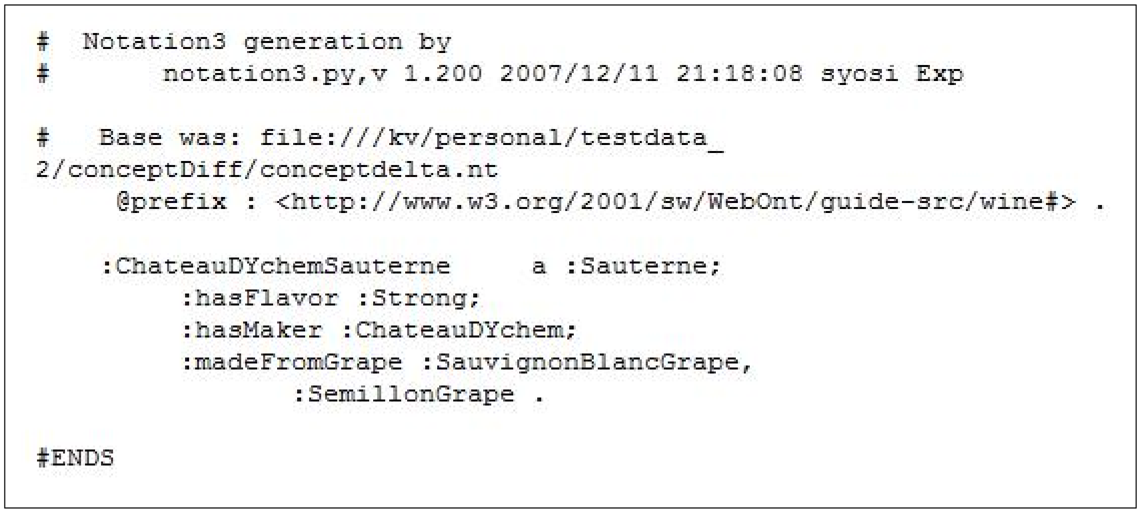
```
#   Base was: file:///kv/personal/testdata_
2/conceptDiff/conceptdelta.nt
     @prefix : <http://www.w3.org/2002/07/owl#> .
    @prefix wine: <http://www.w3.org/TR/2003/PR-owl-guide-
20031209/wine#> .

    wine:Sauternes     a :Class;
         <http://www.w3.org/2000/01/rdf-schema#subClassOf>
wine:Bordeaux,
                wine:LateHarvest,
                 [
             a :Restriction;
             :hasValue wine:SauterneRegion;
             :onProperty wine:locatedIn ],
                 [
             a :Restriction;
             :hasValue wine:Medium;
             :onProperty wine:hasBody ],
                 [
             a :Restriction;
             :hasValue wine:White;
             :onProperty wine:hasColor ] .

#ENDS
```

FIG. A.4. Concept level delta (sauternes-subtractive)



```
#  Notation3 generation by
#       notation3.py,v 1.200 2007/12/11 21:18:08 syosi Exp

#   Base was: file:///kv/personal/testdata_
2/conceptDiff/conceptdelta.nt
     @prefix : <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
.

    :ChateauDYchemSauterne     a :Sauternes;
         :hasFlavor :Strong;
         :hasMaker :ChateauDYchem;
         :madeFromGrape :SauvignonBlancGrape,
                :SemillonGrape .

#ENDS
```

FIG. A.5. Concept level delta (ChateauDYChemSauterne-subtractive)



```
#  Notation3 generation by
#       notation3.py,v 1.200 2007/12/11 21:18:08 syosi Exp

#   Base was: file:///kv/personal/testdata_
2/conceptDiff/conceptdelta.nt
     @prefix : <http://www.w3.org/2001/sw/WebOnt/guide-src/wine#> .

    :ChateauDYchemSauterne     a :Sauterne;
         :hasFlavor :Strong;
         :hasMaker :ChateauDYchem;
         :madeFromGrape :SauvignonBlancGrape,
                :SemillonGrape .

#ENDS
```

FIG. A.6. Concept level delta (ChateauDYChemSauterne-additive)

REFERENCES

[1] Allocca, C.; d'Aquin, M.; and Motta, E. 2009. Detecting different versions of ontologies in large ontology repositories. In *International Workshop on Ontology Dynamic at International Semantic Web Conference*.

[2] Berners-lee, T., and Connolly, D. 2004. D.: Delta: an ontology for the distribution of differences between rdf graphs. http://www.w3.org/designissues/diff. In *RDF Graphs. World Wide Web, http://www.w3.org/ DesignIssues/Diff*, 3.

[3] Brickley, D., and Miller, L. 2010. Foaf vocabulary specification 0.97. Namespace document.

[4] Bunke, H., and Shearer, K. 1998. A graph distance metric based on the maximal common subgraph. *Pattern Recogn. Lett.* 19(3-4):255–259.

[5] Carroll, J. J. 2002. Matching rdf graphs. In Horrocks, I., and Hendler, J. A., eds., *International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, 5–15. Springer.

[6] Carroll, J. J. 2003. Signing rdf graphs. In *In 2nd ISWC, volume 2870 of LNCS*, 5–15. Springer.

[7] Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 380–388. New York, NY, USA: ACM.

[8] Conrad, J. G., and Schriber, C. P. 2004. Constructing a text corpus for inexact duplicate detection. In Sanderson, M.; Järvelin, K.; Allan, J.; and Bruza, P., eds., *SIGIR*, 582–583. ACM.

[9] On the dynamics of linked datasets (esw wiki). http://esw.w3.org/DatasetDynamics.

[10] David, J., and Euzenat, J. 2008. Comparison between ontology distances (preliminary results). In Sheth, A. P.; Staab, S.; Dean, M.; Paolucci, M.; Maynard, D.; Finin, T. W.; and Thirunarayan, K., eds., *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, 245–260. Springer.

[11] Ding, L.; Finin, T.; Joshi, A.; Pan, R.; Cost, R. S.; Peng, Y.; Reddivari, P.; Doshi, V.; and Sachs, J. 2004. Swoogle: a search and metadata engine for the semantic web. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, 652–659. New York, NY, USA: ACM.

[12] Ehrig, M.; Haase, P.; Hefke, M.; and Stojanovic, N. 2005. Similarity for ontologies - a comprehensive framework. In *ECIS*.

[13] Graph update ontology. http://webr3.org/specs/guo/.

[14] Hau, J.; Lee, W.; and Darlington, J. 2005. A semantic similarity measure for semantic web services. In *In: Web Service Semantics Workshop at WWW (2005*.

[15] Heflin, J., and Hendler, J. 2000. Dynamic ontologies on the web. In *In Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000*, 443–449. AAAI/MIT Press.

[16] Heflin, J. 2001. *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment*. Ph.D. Dissertation, University of Maryland, College Park.

[17] Indyk, P., and Motwani, R. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 604–613. ACM New York, NY, USA.

[18] Klein, M., and Fensel, D. 2001. Ontology versioning on the semantic web. In *Stanford University*, 75–91.

[19] Klein, M.; Fensel, D.; Kiryakov, A.; and Ognyanov, D. 2002. Ontology versioning and change detection on the web. In *In 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02*, 197–212.

[20] Leme, L. A. P. P.; Casanova, M. A.; Breitman, K. K.; and Furtado, A. L. 2008. Evaluation of similarity measures and heuristics for simple RDF schema matching. Monografias em Ciłncia da Computao MCC44/08, Department of Informatics Pontifical Catholic University of Rio de Janeiro.

[21] Levi, G. 1973. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo* 9(4):341–352.

[22] Maedche, A., and Staab, S. 2002. Measuring similarity between ontologies. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, 251–263. London, UK: Springer-Verlag.

[23] Manber, U. 1994. Finding similar files in a large file system. In *USENIX WINTER 1994 TECHNICAL CONFERENCE*, 1–10.

[24] Manku, G. S.; Jain, A.; and Sarma, A. D. 2007. Detecting near-duplicates for web crawling. In Williamson, C. L.; Zurko, M. E.; Patel-Schneider, P. F.; and Shenoy, P. J., eds., *WWW*, 141–150. ACM.

[25] Manning, C. D.; Raghavan, P.; and Schütze, H. 2008. *Introduction to Information Retrieval*. New York: Cambridge University Press.

[26] Noy, N.; ; Noy, N. F.; and Musen, M. A. 2002. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *in Eighteenth National Conference on Artificial Intelligence (AAAI-2002*, 744–750.

[27] Papavassiliou, V.; Flouris, G.; Fundulaki, I.; Kotzinos, D.; and Christophides, V. 2009. On detecting high-level changes in rdf/s kbs. In *International Semantic Web Conference*, 473–488.

[28] Radoslaw Oldakowski, C. B. 2005. Semmf: A framework for calculating semantic similarity of objects represented as rdf graphs. In *Poster session of 4th International Semantic Web Conference, Galway, Ireland.*

[29] Rahm, E., and Bernstein, P. A. 2001. A survey of approaches to automatic schema matching. *VLDB JOURNAL* 10:2001.

[30] Sleeman, J., and Finin, T. 2010. A Machine Learning Approach to Linking FOAF Instances. In *Proceedings of the AAAI Spring Symposium on Linked Data Meets Artificial Intelligence*. AAAI Press.

[31] Smith, M. K.; Welty, C.; and McGuinness, D. L. 2004. Owl web ontology language guide. World Wide Web Consortium, Recommendation REC-owl-guide-20040210.

[32] Xiao, C.; Wang, W.; Lin, X.; and Yu, J. X. 2008. Efficient similarity joins for near duplicate detection. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, 131–140. New York, NY, USA: ACM.

[33] Zablith, F. 2009. Evolva: A comprehensive approach to ontology evolution. In Aroyo, L.; Traverso, P.; Ciravegna, F.; Cimiano, P.; Heath, T.; Hyvnen, E.; Mizoguchi, R.; Oren, E.; Sabou, M.; and Simperl, E. P. B., eds., *ESWC*, volume 5554 of *Lecture Notes in Computer Science*, 944–948. Springer.

[34] Zeginis, D.; Tzitzikas, Y.; and Christophides, V. 2007. On the foundations of computing deltas between rdf models. In Aberer, K.; Choi, K.-S.; Noy, N.; Allemang, D.; Lee, K.-I.; Nixon, L. J. B.; Golbeck, J.; Mika, P.; Maynard, D.; Schreiber, G.; and Cudr-Mauroux, P., eds., *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, volume 4825 of *LNCS*, 631–644. Berlin, Heidelberg: Springer Verlag.