Text Based Similarity Metrics and Delta for Semantic Web Graphs

Krishnamurthy Koduvayur Viswanathan and Tim Finin

University of Maryland, Baltimore County
Baltimore, MD 21250, USA
{krishna3, finin}@umbc.edu

Abstract. 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. When one graph is judged to be a version of another, we generate a "delta" consisting of of triples to be added or removed from one graph 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* and evaluated its performance on several tasks using a collection of graphs from the archive of the Swoogle Semantic Web search engine.

Keywords: Semantic Web graphs, similarity metrics, delta

1 Introduction and Motivation

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. [18] present a good survey.

The same problem is 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.

We sought to use the techniques from this body of work, in the Semantic Web, thereby being able to provide similar functions for a Semantic Web search engine. 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 plain text documents. 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. 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.

We explore three different ways in which a pair of Semantic Web graphs can be similar to each other (identified in section 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: there have been a lot of discussions on how linked data consumers could keep track of changes on a Linked Open Data (LOD) dataset[1]. A service like *pingthesemanticweb*[2] 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 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] http://esw.w3.org/DatasetDynamics

[2] http://pingthesemanticweb.com

2 Semantic Web Graph Similarity

The archive of the Swoogle search engine [8] shows several examples of how ontologies and RDF documents evolve over time. For example, there are several copies of the wine ontology [22] found on the web (developed by the W3C's OWL Working Group, and used in the OWL Web Ontology Language Guide). Searching for "wine" on Swoogle returns several results including fourteen 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.

We could also have Semantic Web documents (SWDs) that use the same classes and properties, but contain different instance data in terms of literals e.g. online chat logs from forums or web services that generate FOAF[2] profiles from user entered information. 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 a different serialization format, looks very different in text. We specifically deal with three types of similarity: similarity in classes and properties used while differing only in literal content, difference only in base-URI, and versioning relationship.

3 Related Work

We attempt to identify near duplicate Semantic Web documents that have been created due to revisions, modifications, splitting, copying and merging of documents. Manku et al.[18] present a survey of other near duplicate detection studies. They also describe the design of a near duplicate detection system on a multi-billion page repository. Paes Leme et al. [15] 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.

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. [21] 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. [11] 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 [3] 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 [16]. These methods are not suitable for our use cases. Maedche and Staab [17] 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. [9] describe a framework that aims at comparing concepts across ontologies, and not ontologies themselves. David and Euzenat [7] 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. Carroll [4] explores the equality of two RDF graphs in the light of the graph isomorphism literature.

There are theoretical and experimental studies which deal with ontology versioning. Heflin et al. [12] describe a theoretical model for managing multiple versions of ontologies in a distributed environment. Allocca et al. [1] 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.

With respect to determining the change between two Semantic Web documents, Papavassiliou et al [20] 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. [23] 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 [19] 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. [14] describe the kind of changes possible between two ontology versions and describe a system that provides support for the versioning of online ontologies. The Graph Update Ontology [10] 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 system that would be able to characterize the relationship between two SWDs and automatically generate the diffs between them if they are versions of the same Semantic Web graph. This would require both automatic detection of similarity, and generation of 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 description of how this document is related to the one in the result.

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 1. Two Semantic Web documents before and after canonicalization.

4 Problem Statement and Approach

Allocca et al. [1] define versioning as “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. These variants are created due to the dynamic nature of the web.

Problem Definition: *Given a collection of Semantic Web graphs in the form of RDF documents, identify all pairs of documents that are similar to each other. Characterize the similarity into one or more of the three cases: similarity in classes and properties used while differing only in literal content, difference only in base-URI, and versioning relationship. Further, generate a delta between pairs that have a versioning relationship.*

Our input corpus is in the form of a set of RDF documents. Our approach involves the following steps:

4.1 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. Hence we convert all the documents into a uniform serialization format before comparing them.

4.2 Canonical Representation

Text based comparison methods for SW graphs are affected by blank node identifiers and statement ordering in a document. 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 another graph.

Consider two equivalent Semantic Web graphs (serialized as documents) that use different blank node IDs (see Table 1). Both the graphs are semantically equivalent, but have different representations in text. We use the following algorithm that 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 one-step deterministic labelling algorithm [5].

```
1. foreach triple in graph:
   (a) if triple.subject is a BNode:
       i. $triple.subject \leftarrow " \sim "$
       ii. $triple.meta.subjectcomment \leftarrow triple.subject.nodeID$
   (b) if triple.object is a BNode:
       i. $triple.object \leftarrow " \sim "$
       ii. $triple.meta.objectcomment \leftarrow triple.object.nodeID$
2. Sort all the triples in alphabetical order. (Key for the sort is the text representation of the triple. Ignore the metadata comments)
3. $bTable \leftarrow new\ HashTable$
4. $newID \leftarrow \_ : g1$
5. foreach triple in sorted(graph):
   (a) if triple.meta.objectcomment is not empty:
       i. if bTable not containsKey(triple.meta.objectcomment):
          A. $bTable[triple.meta.objectcomment] \leftarrow$ new BNode(newID)
          B. increment newID
       ii. $triple.meta.object \leftarrow bTable[triple.meta.objectcomment]$
       iii. delete triple.meta.objectcomment
   (b) if triple.meta.subjectcomment is not empty:
       i. if bTable not containsKey(triple.meta.subjectcomment):
          A. $bTable[triple.meta.subjectcomment] \leftarrow$ new BNode(newID)
          B. increment newID
       ii. $triple.meta.subject \leftarrow bTable[triple.meta.subjectcomment]$
       iii. delete triple.meta.subjectcomment
6. Sort all the triples in alphabetical order. Ignore metadata comments
```

In essence, this algorithm first removes all blank node identifiers and replaces them by a special character (tilde). It next sorts all the triples lexicographically, which provides a deterministic ordering to all the triples. Finally, when we encounter a blank node identifier, we look up our *bTable* table to see if we have already created a new identifier for it. If so, we replace it by the new identifier. Otherwise, we create a new identifier in numerical order and make a new entry into the table before replacing the old identifier with the new identifier.

As can be seen from Table 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.

Limitation of the Algorithm: Non-Distinctive Triples. The algorithm is able to correctly rename blank nodes for only those triples, that can be uniquely identified in the graph even after all blank nodes are treated as equal. Such triples are called non-distinctive triples[5]. A group of non-distinctive triples appears the same regardless of the relative ordering of the triples in the group.

For a group of n non-distinct triples, there are $n!$ relative orderings of the triples. Hence there are $n!$ ways of renaming the blank nodes. Additionally, 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. We determined the number of possible canonical forms for each SW document in a collection of 1200 randomly selected RDF documents from Swoogle. The results (as shown in Fig 1) indicate that only 13% of the RDF documents in the collection don't have a unique canonical form and hence cannot be processed by our system.

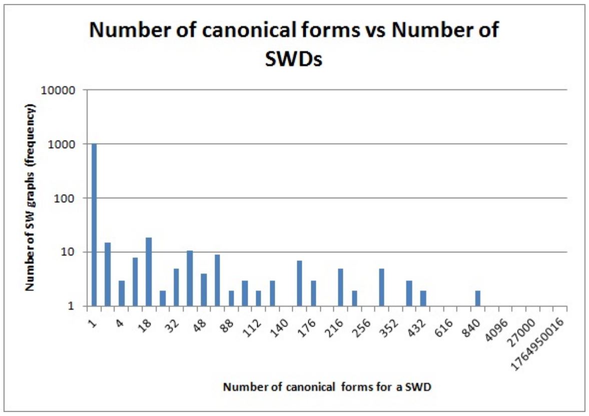

Fig. 1. Number of Canonical Forms for SW Graphs

If blank nodes are in triples with functional or inverse function properties it may be possible to deduce that two blank nodes must refer to the same object or distinct objects. For example, assume that *ex:fp* is an *owl:functionalProperty* and *ex:ifp* is an *owl:inverseFunctionalProperty* and we have the two graphs shown in Figure 2. 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*.

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

Fig. 2. Functional and inverseFunctional properties can be used to deduce that two blank nodes are owl:sameAs or owl:differentThan.

4.3 Reduced Forms

In order to detect the various aspects of similarity, the original graphs are decomposed into forms where these can be detected by transforming the graphs triplesto produce a serializable form. We generate the following four reduced forms from the *canonicalized n-triples form* (section 4.2):

1. A document containing 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.
2. A document with 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.
3. A document that has the base-URI of every node replaced by the empty string. This form lets us compare only the local names of the classes and properties used in both the graphs. This form includes the literal content of the graph.
4. A document where all the literals and the base-URI of every node are 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 Semantic web graphs with respect to their non-literal content.

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

In our study, all 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. For a definition of similarity measures, refer to [15].

We use the following measures:

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:

$$jaccard(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|} \tag{1}$$

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

Containment of set A in set B is defined as:

$$containment(A, B) = \frac{|S(A) \cap S(B)|}{|S(A)|} \tag{2}$$

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

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 n-grams. These sets are then used to compute the measures as indicated above.

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 $\mathit{\Gamma}$ is the set of terms appearing in these SWDs, then a Semantic Web document vector containing the terms $\mathit{\Gamma}_j = (t_1, t_2, ..., t_n)$ 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 $t_i \in \mathit{\Gamma}_j$.

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:

$$similarity(A, B) = cos(\theta_{A,B}) = \frac{\overrightarrow{A}.\overrightarrow{B}}{\|\overrightarrow{A}\|\|\overrightarrow{B}\|} \tag{3}$$

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

Charikar's Simhash : This fingerprinting technique was developed by Charikar [6]. It is a Locality Sensitive Hashing [13] method which has a property that simhash fingerprints of similar documents are mapped close together, i.e. they differ in a small number of bit positions. 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 pre-determined 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 (refer to [7]), instead of a similarity metric. 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 $n >> k$.

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.

4.4 Pairwise Computation of Metrics

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:

4.5 Similarity Metrics

1. Compute the two cosine similarity values between the canonical representations (already generated) of both the SWDs.
2. If the cosine similarity values are below a pre-determined threshold, then eliminate this pair from further consideration, else add this pair to a list of candidate pairs.
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 initial filter to reduce the remaining computations.

4.6 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 a dataset is annotated with the ground truths about whether the SWDs in a given pair in the dataset are similar to each other (refer to Section 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. 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. These classifiers are then used to detect the various forms of similarity that we have defined.

4.7 Computing Delta Between Two Versions

Once it is determined that two SWDs have a versioning relationship between them, we compute the set of statements that describe the change between successive versions of the SWD. As a result, we use the concept of additive and subtractive deltas. The *additive delta* is the set of triples that are added to the older version in order to generate the newer version. The *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 four techniques to describe the change.

Raw delta. This method simply computes a raw delta by doing a statement-by-statement comparison between the two SWDs. 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 is considered, i.e. we ignore the base-URI of all entities. It is possible to compute the delta using this method because the canonicalization process that we apply on each SWD smooths the disparities between statements in similar documents. It assigns unique and standard identifiers to blank nodes and deterministically orders the statements

Delta of deductive closures. This method first computes the deductive closures of the two, and then the resultant graphs are converted to their canonicalized form using the canonicalization algorithm. Then a raw delta is generated between them.

Delta at class level of an ontology. This method applies only to ontology version pairs that are serialized in the form of XML. 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. This approach is a variant of the one followed by [14]:

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 a concept in the ontology.
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. These two graphs can now be compared at a statement level as done before for the raw delta.

Heuristic methods. Sometimes several statements are generated in a delta as a result of a relatively simple conceptual change like the renaming of a class. We detect such a class renaming. Essentially, the 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 is actually an instance of a class renaming. Similarly we check for renaming of object-classes.

5 Evaluation

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 exists no standard labeled dataset of similar Semantic Web documents that we could use for the purpose of evaluating our system. Hence we constructed a collection of Semantic Web documents from Swoogle's Semantic Web archive and cache services[4]. Swoogle periodically crawls the Semantic Web and maintains several snapshots for each indexed SWD. We added such versions to our data-set and labeled them as having a versioning relationship.

5.1 Detecting Pairs that Differ only in Literal Content

As described in section 2, SWDs can share the same classes and properties while being different only in the literal content used. To detect this kind of similarity, we build a feature vector using the following measures: CosineSim, LocalNameNoLiteralJaccard, and LocalNameNoLiteralSimhash. (refer to 4.4)

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 labeled such pairs as positive. 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 Table 2

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

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

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

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

.

5.2 Detecting Pairs that Differ Only in Base-URI

Section 2 described examples of Semantic Web documents that differ only in the base-URI. In order to detect similarity of this kind between a candidate pair, the metrics we use are CosineSim, LocalNameCosineSim, LocalNameNoLiteralJaccard, LocalNameNoLiteralContainment, OnlyLiteralContainment, OnlyLiteral-Jaccard.

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 5.1. 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

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 Table 3. A Support Vector Machine (linear kernel) trained using the same set of feature vectors did slightly worse than the Naive-Bayes classifier.

We tried this classifier on a set of ontologies comprising the wine[5], baseball[6], geospecies[7] , and dbpedia[8] ontologies to ensure that this classifier can identify cases that occur in the real world. The classifier successfully identified each pair.

5.3 Detecting Pairs with a Versioning Relationship

As mentioned already, we used the Swoogle cache to get snapshots of Semantic Web documents at different instances of time.

Our system cannot determine a versioning relationship between the snapshots of SWDs that are highly dynamic over time. But the other pairs that have a relatively smaller amount of change between them can actually be considered as having a versioning 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 containing 160 instances, (50% positive, and 50% negative) that was built the same way as the training data-set. The results of the classification are as shown in Table 4.

6 Discussion

Zeginis et al. [23] define the following notion of correctness for computing the delta of two RDF graphs. 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.

Limitations. We currently do not have a principled filtering mechanism that will reduce the number of comparisons from the current quadratic scale. Hence we are not able to perform these computations a for large collection of Semantic Web documents. The computation method is shallow, i.e. at the text level, but it is suitable for our use cases. The other graph based and relational-instance set similarity measures mentioned in the related works section are very expensive for such cases.

[5] http://w3.org/2001/sw/WebOnt/guide-src/wine

[6] http://www.damn.org/2001/08/baseball/baseball-ont

[7] http://rdf.geospecies.org/ont/gsontology

[8] http://downloads.dbpedia.org/3.2/en/dbpedia-ontology.owl

7 Conclusion and Future Work

We explored some of the ways in which Semantic Web graphs may be similar to each other. We developed a system *Simils* that can recognize when two Semantic Web graphs are similar and characterizes their difference in three ways. The tool can be used with the pre-trained classifier, but it is preferable for users to re-train the classifier on their own datasets. 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 then used to compute text based similarity metrics between candidate pairs. These similarity metrics are then used to characterized the candidate pair as mentioned above. Further, when the system detects a versioning relationship between a pair, it 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 whether the statements in the delta between two ontologies can be accounted for by a relatively simple conceptual change like the renaming of a class.

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. One of the future directions might be to implement a principled filtering mechanism that would reduce the number of comparisons, so that we can increase scalability. The deltas that we generate are in the form of n3. We would like to use a canonicalized ontology for representing the comparison between two candidate Semantic Web documents: including the description of the type(s) of similarity between them, and the delta between them (if applicable). Another direction for future work would be to develop a way of guaranteeing a small-sized delta.

References

1. C. Allocca, M. d'Aquin, and E. Motta. Detecting different versions of ontologies in large ontology repositories. In *Int. Workshop on Ontology Dynamic at Int. Semantic Web Conf.*, 2009.
2. D. Brickley and L. Miller. Foaf vocabulary specification 0.97. Namespace document, January 2010.
3. H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recogn. Lett.*, 19(3-4):255–259, 1998.
4. J. J. Carroll. Matching rdf graphs. In I. Horrocks and J. A. Hendler, editors, *Proc. 1st Int. Semantic Web Conf.*, volume 2342 of *Lecture Notes in Computer Science*, pages 5–15. Springer, 2002.
5. J. J. Carroll. Signing rdf graphs. In *In 2nd ISWC, volume 2870 of LNCS*, pages 5–15. Springer, 2003.
6. M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proc. of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, New York, NY, USA, 2002. ACM.

7. J. David and J. Euzenat. Comparison between ontology distances (preliminary results). In A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, editors, *Int. Semantic Web Conf.*, volume 5318 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2008.
8. L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM '04: Proc. of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM.
9. M. Ehrig, P. Haase, M. Hefke, and N. Stojanovic. Similarity for ontologies - a comprehensive framework. In *ECIS*, 2005.
10. Graph update ontology. http://webr3.org/specs/guo/.
11. J. Hau, W. Lee, and J. Darlington. A semantic similarity measure for semantic web services. In *In: Web Service Semantics Workshop at WWW (2005*, 2005.
12. J. Heflin and J. Hendler. Dynamic ontologies on the web. In *In Proc. of the Seventeenth National Conf. on Artificial Intelligence (AAAI-2000*, pages 443–449. AAAI/MIT Press, 2000.
13. P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM New York, NY, USA, 1998.
14. M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. Ontology versioning and change detection on the web. In *In 13th Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW02*, pages 197–212, 2002.
15. L. A. P. P. Leme, M. A. Casanova, K. K. Breitman, and A. L. Furtado. Evaluation of similarity measures and heuristics for simple RDF schema matching. Monografias em Cincia da Computao MCC44/08, Department of Informatics Pontifical Catholic University of Rio de Janeiro, Oct. 2008.
16. G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341–352, 1973.
17. A. Maedche and S. Staab. Measuring similarity between ontologies. In *EKAW '02: Proc. of the 13th Int. Conf. on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 251–263, London, UK, 2002. Springer-Verlag.
18. G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for web crawling. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *WWW*, pages 141–150. ACM, 2007.
19. N. Noy, , N. F. Noy, and M. A. Musen. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *in Eighteenth National Conf. on Artificial Intelligence (AAAI-2002*, pages 744–750, 2002.
20. V. Papavassiliou, G. Flouris, I. Fundulaki, D. Kotzinos, and V. Christophides. On detecting high-level changes in rdf/s kbs. In *Int. Semantic Web Conf.*, pages 473–488, 2009.
21. C. B. Radoslaw Oldakowski. Semmf: A framework for calculating semantic similarity of objects represented as rdf graphs. In *Poster session of 4th Int. Semantic Web Conf., Galway, Ireland*, 2005.
22. M. K. Smith, C. Welty, and D. L. McGuinness. Owl web ontology language guide. World Wide Web Consortium, Recommendation REC-owl-guide-20040210, February 2004.
23. D. Zeginis, Y. Tzitzikas, and V. Christophides. On the foundations of computing deltas between rdf models. In *Proc. of the (ISWC/ASWC2007), Busan, South Korea*, LNCS, pages 631–644. Springer Verlag, 2007.