A Performance Analysis Framework for Coreference Resolution Algorithms

Chandankumar Johakhim Patel
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Sciences Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/1569

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu, library-corescholar@wright.edu.
A Performance Analysis Framework for Coreference Resolution Algorithms

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Science

By

CHANDANKUMAR JOHAKHIM PATEL
Wright State University, 2016

2016
Wright State University

Michelle A. Cheatham, Ph.D.
Thesis Director

Mateen M. Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

Michelle A. Cheatham, Ph.D.

Mateen M. Rizki, Ph.D.

Tanvi Banerjee, Ph.D.

Robert E.W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School
ABSTRACT


This thesis entitled A Performance Analysis Framework for Coreference Resolution Algorithms, focuses on the topic of coreference resolution of semantic datasets. In order for Big Data analytics to be effective, it is essential to develop automated algorithms capable of integrating multiple datasets that contain data about a particular person or other entity. Accomplishing this necessitates coreference resolution; for example, determining that J. Doe in one dataset refers to the same person as Jonathan Doe Jr. in another dataset. There are many existing coreference resolution algorithms, but there are only a few basic design decisions to be made by such systems when it comes to how to compare two individual instances. An analysis framework is presented that assesses the impact of different choices for these design decisions on coreference resolution in terms of precision, recall, and F-measure.
# Table of Contents

1. Introduction.................................................................................................................. 1

1.1. The Four Principles of Linked Data........................................................................ 2

1.1.1. First Principle....................................................................................................... 2

1.1.2. Second Principle ................................................................................................. 3

1.1.3. Third Principle .................................................................................................... 4

1.1.4. Fourth Principle .................................................................................................. 5

1.2. Coreference Resolution............................................................................................. 7

1.3. Goals .......................................................................................................................... 13

2. Literature Review......................................................................................................... 15

2.1. EXONA....................................................................................................................... 15

2.2. InsMT+ ...................................................................................................................... 16

2.3. Lily ............................................................................................................................ 17

2.4. LogMap...................................................................................................................... 18

2.5. RiMOM....................................................................................................................... 19

2.6. STRIM ....................................................................................................................... 20

2.7. SLINT++................................................................................................................... 20

2.8. SBUEI ....................................................................................................................... 21

2.9. AgreementMaker .................................................................................................... 22
3.2.1. Input .......................................................... 34

System .............................................................. 37

3.2.2. Output ......................................................... 38

3.3. System Operation ............................................ 40

4. Results and Analysis ........................................ 45

4.1. Impact of the Information Used For Comparison ........ 45

4.2. Impact of the String Similarity Metric .................. 48

5. Conclusions and Future Work ............................... 50

5.1. Conclusions ..................................................... 50

5.2. Future Work .................................................... 52

6. References ........................................................ 54

Appendix A: UML Diagrams ..................................... 57
Figure 1: One representation of Barbara Hickey and related entities ......................... 9
Figure 2: A second representation of Barbara Hickey and related entities ............. 10
Figure 3: Architectural Overview of the System ................................................. 34
Figure 4: Concept of Precision and Recall ...................................................... 39
Figure 5: Sample Output Graph ..................................................................... 40
Figure 6: Selecting Datasets .......................................................................... 41
Figure 7: Dialog box to Select Dataset .......................................................... 41
Figure 8: Similarity Algorithms ..................................................................... 41
Figure 9: Selecting Thresholds for Non-parametric Comparison ...................... 42
Figure 10: Selecting Various Accuracies for Parametric Comparison ............... 42
Figure 11: Tabular Result of Matches Found .................................................. 43
Figure 12: Available Options to Vary Various Parameters ............................... 43
Figure 13: Selecting Reference Alignment File ............................................... 44
Figure 14 Impact of the Levenshtein metric .................................................. 49
Figure 15 Impact of the JaroWinkler metric .................................................... 49
Figure 16 Impact of the LCS metric ................................................................. 49
# List of Tables

Table 1: Comparison Methodology of Current Coreference Resolution Systems ..... 30
Table 2: UML for the OntologyMatcher Class ......................................................... 57
Table 3: UML for the Ontologies Class ................................................................. 58
Table 4: UML for the MatchedInstanceData Class .............................................. 58
Table 5: UML for the LCS (Longest Common Substring) Class ....................... 59
Table 6: UML for the Jaro Winkler Class .............................................................. 59
Table 7: UML for the IterativeMatching Class ...................................................... 59
Table 8: UML for the InstanceData Class ............................................................. 60
Table 9: UML for the BasicUtilities Class ............................................................. 60
LIST OF ALGORITHMS

Algorithm 1: Get Ontology List

Algorithm 2: Levenshtein Similarity
ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to my thesis advisor Dr. Michelle A. Cheatham, for guiding me through this endeavor and being a source of constant inspiration. Despite battling with thoroughly preoccupied and strenuously tight schedules, you’ve spared time to render every possible effort to help me go through this thesis work, and provided infrastructure and resources as well. Ma’am, thanks a million for being lapis philosophorum!!
Chapter 1

1. Introduction

Through the Internet there is now more data available to all of us than ever before, but this data is unfortunately not as useful as it could be. While individual facts are important, knowledge is only gained when it is possible to relate facts to one another in order to understand the “big picture.” For instance, one database may contain information about principal investigators on NSF grants, while a separate online catalog may contain a list of collections of measurements gathered by various scientists and how often each collection has been downloaded. If these two datasets could be accessed, queried, and analyzed in a uniform manner, then many interesting and important observations become possible, such as determining which NSF funding awards have led to measurements that have been useful to the largest number of researchers. However, several obstacles stand in the way of this type of seamless exploration across datasets.

One issue is that the enormous amount of data on the web is available in a wide variety of forms and formats and cannot be accessed in a consistent and unified manner. Attempts to address this obstacle to data integration have led to the rise of linked data, which was originally proposed by Tim Berners-Lee as a key component of the Semantic Web. According to Berners-Lee, the Semantic Web is not concerned only with putting data on the web, but is also about establishing relationships so that
both humans and machines can access the data. In other words, linked data is about establishing relationship links between data so as to make it accessible. Just like a link on the webpage, which connects hypertext documents, linked data triples encode links between random data expressed in RDF – Resource Description Format.

1.1. The Four Principles of Linked Data

The four principles of linked data are [Berners-Lee et. al. 2001]

• Use URIs as names for things
• Use HTTP URIs so that people can look up those names.
• When someone looks up a URI, provide useful information, using the standards (e.g. RDF, SPARQL)
• Include links to other URIs, so that they can discover more things.

These linked data principles provide an infrastructure to share data from different sources. Linked data can be thought of as a formal specification set, following which data across the Internet can be retrieved, shared and processed.

1.1.1. First Principle

The first principle of linked data, “use URIs as names for things” often causes some confusion, because many people do not fully understand the relationship between terms such as URL, URN, URI, and IRI. Typically, to access any resource on internet you need to know the server address, the directory where the document is located and the resource name itself; e.g. http://www.chandanpatel.com/certificates/transcript.html is the necessary combination of server address – www.chandanpatel.com, directory – certificate, and the name of the resource – transcript.html. Tim Berners-Lee coined the term Uniform
Resource Locator – **URL** for this server address – directory – resource name combination. Generally people believe that anything starting with http:// is an address for some webpage that they can see in browser – but that is not true. The reason is, if one knows the directory structure on the server, a reference can still be made to a server address – directory – resource combination that is not actually a web page, but some sort of other resource. Along this same line of thinking, two engineers, one from MIT and the other from Xerox, developed a system to name resources called Uniform Resource Name – **URN**. A URN uniquely describes a resource, e.g. `urn:chandanpatel.com:essays:StatementOfPurpose` would uniquely describe the statement of purpose; or `urn:isbn:978-1-11884571-1` may describe the book *Beginning Visual C++ 2013* by Ivor Horton. **URI** stands for Uniform Resource Identifier. This term was meant to cover both URLs and URNs. Because the use of URNs is very small compared to that of URLs, the term URI is sometimes conflated with URL, but this is not technically correct – a URL is also a URI, but the converse may not be true. Finally, the term **IRI**, which was developed by the Internet Engineering Task force, stands for Internationalized Resource Identifier. Basically, an IRI is a URI that allows the use of characters from languages other than English. For describing naming resources in this thesis I shall refer to IRIs.

### 1.1.2. Second Principle

The second linked data principle is to use HTTP URIs so that people can look up things they are interested in. Basically the web is a client-server mechanism, and in this mechanism, the client and the server need to talk to each other; and this requires a clear-cut specification so that each can understand the other. This is
achieved through a protocol – there has to be a well specified way in which a client can send a request to a server, as well for the server to send the requested resource back to the client (or denying the request, sending an error message, etc.). This requirement is achieved by using the Hyper Text Transfer Protocol i.e. HTTP.

1.1.3. Third Principle

The third linked data principle is to provide useful information when someone looks up a URI, using the appropriate standards. On the traditional web, documents are usually described using Hyper Text Markup Language – HTML. HTML is not appropriate for describing information on the Semantic Web, because it is more concerned with describing how a document should be formatted than how the information contained on that document relates to other information. For example, typical HTML tags used within a document control the size, font, and color of text, along with its placement on the page. Thus, HTML makes data readable by humans but is not helpful for machines, which violates the linked data principles. Instead, the Resource Description Framework – RDF – is generally used on the Semantic Web, either alone or in conjunction with HTML. An RDF statement is a triplet of subject, predicate and object, similar to a simple sentence. For example, the simple statement Chandan’s cellphone number is 908-627-1897 can be represented as a data triplet as shown below:

<domain1/cPatel> <foaf:phone> "908-627-1897"@en .

Similarly, the statement Chandan’s preferred internship field is software development can be represented as:

<domain2/chandanPatel>
In the first of these examples the subject is a URI representing Chandan – domain1/cPatel. Anyone publishing linked data needs to acquire a domain name such as “domain1” and establish a procedure for minting unique URIs to represent things. Following the subject is a URI representing the predicate (relationship) we wish to address – foaf:phone. This URI can be chosen by the data publisher, but in this case an existing vocabulary of terms is being used. FOAF\(^1\) is an acronym for Friend of a Friend. It is an ontology used to describe people and the relationships between them. Finally, the first statement ends with a literal value – Chandan’s phone number.

The second example differs slightly from the first in that the object of the RDF triple is not a simple literal value in this case, but rather another URI that represents a complex entity. Predicates such as foaf:phone, which have a complex entity as a subject but a literal value as an object, are called *datatype properties* whereas predicates such as AcademIS:hasPreferredInternshipField, which have complex entities as both the subject and object, are called *object properties*.

### 1.1.4. Fourth Principle

The last linked data principle is to include links to other URIs that are related to the one the person has looked up. In the example above, the second statement contains a URI representing the field of software development. Looking up this URI would provide more information about that field. This aspect of linked data is critical to contextualizing information and thereby making it more useful.

---

\(^1\) [http://xmlns.com/foaf/spec/](http://xmlns.com/foaf/spec/)
One problem may be noticed in the example above: both linked data publishers have minted their own URIs, so two separate identifiers – domain1/cPatel and domain2/chandanPatel – refer to the same person. This is a problem because, for example, if a recruiter for a software development company wanted to call students regarding potential internships, they would be able to query this data and discover that domain2/chandanPatel is interested in software development internships, but there would be no way to retrieve the phone number because there is no way to connect domain2/chandanPatel to domain1/cPatel. What is needed is a triple of the form:

\[ \text{<domain1/cPatel> <owl:sameAs> <domain2/chandanPatel>} \]

But who should create these links between the same entities in different datasets? This is a tricky question in practice. Datasets are often published completely independently, by researchers who have no connection to (or even knowledge of) one another. What is needed is an automated system that is capable of establishing these links without manual intervention. This is the goal of coreference resolution systems. Specifically, coreference resolution is the activity of determining all of the references made to an entity in RDF / OWL files under consideration so as to highlight the existing relationships between these two datasets about that particular entity, or to establish a relationship between similar entities in two different datasets. Typically, coreference resolution algorithms are applied in the following areas:

1. Similarity based query processing (also called approximate querying). Here one tries to locate all of the data instances that represent the entity indicated in the query.
2. Data integration, where one tries to identify all manifestations of one object in different data sources.
3. Data cleaning, where one tries to find and remove the errors / mistakes from data received from different sources, particularly those related to duplication and misspellings.

1.2. Coreference Resolution

Coreference resolution is a very difficult problem. Obviously, the source of linked data is the web, and data on the web is produced at different times, by different people, who often have strikingly different perspectives. It is but natural to expect in this scenario that the same entity is most probably described in different ways and in different measures within diverse datasets. These differences can arise in data structure as well as in data values.\[^2\] In coreference resolution the focus is on how to establish a connection between two different datasets with respect to an individual (as opposed to a conceptual schema entity); i.e. what is the likelihood that two references in different datasets refer to the same individual. If this likelihood exceeds some threshold value, a “sameAs” or other type of equivalence relation can be established between the instances.

Traditional coreference resolution algorithms basically rely on three design decisions (a) What information should be used to compare two individuals, (b) How should this information be numerically evaluated and (c) whether or not the results of this numerical evaluation should be considered a match. When comparing two instances from two different datasets, the information used to compare them may

include the content of the instance (i.e. the property values), or the structure of the instance (i.e. the existence of properties), or both. And how to numerically evaluate this information depends on whether one is measuring the similarities i.e. nearness of the individuals to each other or distance i.e. level of difference between them. These values are usually based on string metrics, such as Levenstein or Jaro Winkler. After computing this numeric similarity or difference value, a threshold is often used to determine whether or not these two individuals should be considered the same. A possible fourth decision is which individuals should be compared to each other. The simplest coreference resolution algorithms compare all individuals from one dataset to all of those in the other, but this is not scalable for large datasets, so sometimes filtering methods can be used to reduce the number of comparisons.
Figure 1: One representation of Barbara Hickey and related entities
For example, Figures 1 and 2 show information from two different datasets from the GeoLink\(^3\) knowledge base. GeoLink is a National Science Foundation project tasked with integrating seven of the largest oceanographic datasets in the United States according to the linked data principles. Both of the figures shown are making a reference to the same person, Dr. Barbara M. Hickey, but in a different way. A

\(^3\) [http://www.geolink.org/](http://www.geolink.org/)
coreference resolution algorithm that considers property values (design decision a) collects all of the values related to the first instance and all of those related to the second instance. In this case those sets would be:

instance 1: (Dr, Barbara, M, Hickey, University of Washington (UW), scientist-chief, 2006-05-21)

instance 2: (Barbara, Hickey, scientist, University of Washington, mb58+wcd, http://get.rvdata.us/cruise/TN281/fileset/105245)

Then a set similarity metric, such as Jaccard, is computed on these two sets. The Jaccard metric is the intersection of the two sets divided by the sum of the sizes of the sets. To compute this value, there must be some way to decide if two items in the sets are the same. This is design decision b. One way is to use a string similarity metric like Levenstein. This metrics considers how many additions, substitutions, or deletions must be done to change one string into another. If this number is less than some threshold value, the items are considered the same. In some cases using a string similarity metric leads to poor results. For example, if two dates 2006-12-31 and 2007-01-01 are compared as strings, they are not very similar. But these dates are actually very close. Another way to compare property values is to determine the datatype of the values and to use a similarity metric that is appropriate for that type. In this example, the values Barbara/Barbara, Hickey/Hickey, and University of Washington/University of Washington (UW) are common to both sets and would likely be above the threshold for the similarity metric. While scientist/scientist-chief are similar, they would not likely be above the threshold for most string similarity
metrics. In this case, the similarity between the two sets is \( \frac{3}{7 + 6} = 0.23 \). If this is above the similarity threshold (design decision c), then these two instances will be considered a match.

In the approach described above, the parameter values are considered as a “bag of words,” meaning that if a word in the first set is similar to any word in the second set, even if those values are for different properties, they would be considered an indication of similarity. For example, if Barbara Hickey were compared to a person named Ralph Barbara, these people would be considered somewhat similar even though the value they have in common are for different parameters (given name versus family name). To prevent issues like this, it is possible to consider parameter names as well as parameter values when comparing two instances. For example, the two instances above would be compared like this:

instance 1: (namePrefix->Dr, nameGiven->Barbara, nameMiddle->M, nameFamily->Hickey, hasAffiliation->label->University of Washington (UW), isPerformedBy->hasRoleType->scientist-chief, isPerformedBy->hasParticipant->hasTimestamp->2006-05-21)

instance 2: (hasNameGiven->Barbara, hasNameFamily->Hickey, RoleType->label->scientist, Organization->label->University of Washington, hasDataset->hasFormatType->label->mb58+wcd, hasDataset->source->http://get.rvdata.us/cruise/TN281/fileset/105245)

Two values would then match only if both their property name and value matched. Matching of property names can be done with a basic string similarity metric or a more advanced ontology alignment system.
In this example, the names of the instances themselves (Person_105245 and Person_58015) are not meaningful. This is the result of the URI minting strategy chosen by the GeoLink dataset providers. However, some datasets give the instances meaningful names, such as Barbara_Hickey or Hickey_Barabara_M. In such cases, a coreference resolution algorithm may also compare the URIs of the instances directly as part of the similarity computation. Alternatively, some coreference resolution approaches compare instances directly based on their syntactic labels. This in essence is comparing the individuals based on a single property value – rdfs:label or the equivalent.

1.3. Goals

One thing that becomes clear when reading about existing work on coreference resolution is that many algorithms for this purpose are evaluated only on their overall effectiveness. It is not clear which aspects of the algorithms are contributing the most to this overall performance. The goal of this work is therefore to develop an analysis framework that is capable of analyzing the trend in performance metrics based on the choices related to the primary design decisions of a coreference resolution algorithm: (a) What information is used to compare two individuals (property name, property value, or both), (b) How this information is numerically evaluated (comparing all values as strings, or using datatype-specific comparisons) and (c) whether or not the results of this numerical evaluation should be considered a match (by varying the threshold value).
The most established benchmark for evaluating the performance of coreference resolution algorithms is the instance matching track within the Ontology Alignment Evaluation Initiative\(^4\) (OAEI). Because this benchmark is used very often, there is a danger that some coreference resolution algorithms might over fit this benchmark, meaning that they perform well on the instance matching tasks in the benchmark but poorly on real-world instance matching tasks. Therefore, the system to be developed here needs to be able to read in a wide variety of ontologies in order to test the performance of a coreference resolution algorithm. These formats include Web Ontology Language (.owl) files, Resource Description Framework (.rdf) files, and Turtle (.ttl) files. The analysis framework must be also be able to read in a reference alignment (i.e. a set of correct coreference) and compute performance in terms of number of matches found, F-measure, precision, and recall. Ideally, the analysis framework should be able to compute the trend in these metrics for different threshold values.

\(^4\) [http://oaei.ontologymatching.org/](http://oaei.ontologymatching.org/)
2. Literature Review

The Ontology Alignment Evaluation Initiative (OAEI) has become the foremost venue for researchers focused on ontology alignment and coreference resolution to showcase their work. The OAEI introduced an instance matching track in 2009. Every year since then, three to six coreference resolution systems have participated in the track. There have been a total of twenty unique systems. System developers are asked to submit a paper describing the operation of their algorithm when they participate in the OAEI. I have reviewed each of these papers in order to determine the techniques that are common to many different systems. In cases where the same system has participated in multiple years, the review is based on the description of the system during the most recent year it participated. Several systems did not submit papers, and so they have been omitted from this review. After reviewing the approach taken by each coreference resolution system, some common themes are presented and a table showing each system’s approach to comparing two individuals with regards to the topic of this thesis (comparison of property values, property names, or both and whether or not the comparisons are datatype-specific) is presented.

2.1. EXONA

EXONA [Damak 2015] has three modules, namely, transformation, indexation and correspondence, and each module has two phases. During the first phase of the transformation stage, EXONA reads in two OWL files and converts them into
graphs, while in the second phase it creates an *instance object* from the identifier and the contents uniquely identified by each URI. The content has a list of neighboring instances within a certain level of similarity as obtained by an edit distance metric. The *indexation stage* also has two parts, called pretreatment and indexation. In the pretreatment phase, stop words and symbols are removed, while indexation is done using the unique combination of URI and data property value. *Correspondence* is the third module, and it also has two phases, namely *querying* and *filtering and match identification*. During the querying phase, an instance in chosen and a score is obtained by comparing it with each instance in the target dataset. In the filtering and match identification stage, terminological similarity for the two instances with the highest score from the querying phase is calculated. This pair is included in the alignment if the terminological similarity is above a threshold value. This is a graph based matching approach. It gives reasonably good results on the OAEI sandbox task (a task with a smaller dataset) but does not perform well with large scale datasets (the OAEI “Mainbox” task), which may be attributed to the indexing approach taken by the authors.

### 2.2. InsMT+

InsMT+ [KHIAT, A., & BENAISSA, M. 2015] stands for *Instance Matching at the Terminological Level*. In its first step, the system gathers and normalizes the labels of the concepts and properties related to each of the instances. To accomplish this InsMT+ does some preprocessing like converting the case of all words to either lower case or upper case and removing stop words. In the second step it calculates the
similarity between these sets for each pair of instances using various lexical string similarity metrics, Levenshtein, Jaro and the SLIM-Winkler algorithm. The results from these metrics are each stored in a separate results matrix. In the third stage, a local threshold value is applied to the results in each of these results matrices in order to filter out unlikely coreference. Afterwards, the individual metric results matrices are aggregated together into a single results matrix, and a global threshold is applied to this combined matrix in order to generate the final matches. Thus InsMT+ applies two sets of filters (thresholds) – first, to each individual matrix for each individual string metric used, and second, the combined matrix with the aggregated (averaged) results of the individual matrices. My view is that, because this system applies filters at two different stages and averages the results of many different similarity metrics, its performance as reflected in its precision and recall values becomes more symmetrical about the mean when compared to each metric alone. With smaller datasets InsMT+ can sometimes achieve better accuracy, but with larger datasets this symmetry effect becomes more noticeable. Hence the performance of this system in only average.

2.3. Lily

Lily [WANG, W., & WANG, P 2015] has different modes of operation for different types of tasks, for example for common matching tasks it uses GOM – Generic Ontology Matching, while for large datasets it uses LOM – Large-scale Ontology Matching. Lily’s module for instance matching is called IOM. Lily’s matching process is divided into three stages. In the first stage the ontologies are
prepared for input to the system, in second phase the similarity is computed using a special algorithm (described below), and the mapping results are refined in the third stage. Lily uses semantic subgraphs to obtain possible meanings of the ontology elements by creating a hybrid ontology graph that represents the semantic relations between elements. This graph is based on an electrical circuit model. This algorithm generates a Semantic Description Document that contains information on class hierarchies, domain and range of properties, and other aspects of the entities. Then this Semantic Description Document is used along with text matching and structural matching techniques to calculate the similarity between entities. To handle large scale ontologies Lily uses the concept of reduction anchors – positive reduction anchors use the concept hierarchy to predict ignorable similarities and negative reduction anchors use the locality of matching to predict ignorable similarities. With this strategy the system yielded very high precision and recall values on many OAEI tracks. Furthermore, the system works with nearly equal efficiency with data of small size, e.g. on the sandbox task, and with larger ontologies, as in the Mainbox task.

2.4. LogMap

LogMap is a scalable alignment system and has features like lexical indexation, logic based module extraction, propositional Horn reasoning, axiom tracking and semantic indexation. Lexical indexation allows LogMap to use lexical information, such as entity labels, from the input ontologies. The logic-based module enables the system to determine when two entities cannot possibly match. Un-satisfiable classes can be detected using propositional reasoning. Matches are then removed strategically
in order to resolve the inconsistencies in the alignment. The system’s high scalability with respect to detecting logical inconsistencies relies on an extension of the Dowling–Gallier algorithm to track all mappings involved in each logical inference. Though it is capable of coreference resolution, LogMap is primarily an ontology alignment system focused on generating schema alignments that exhibit logical coherence, i.e. that do not have any logical conflicts as evidenced by unsatisfiable classes.

2.5. RiMOM

RiMOM [ZHANG, Y., & LI, J. 2015] is divided into several modules, namely, Preprocessing, Predicate Alignment, Choosing a Matcher, Generating Candidate Pairs, Matching Score Calculation, Instance Alignment and Validation. In the preprocessing step stop words are removed. In the predicate alignment step, one-to-one relationships are found between properties using the Jaccard similarity metric. For each instance in the source ontology, the candidate pair generation stage selects instances in the target ontology as potential matches only if it shares a common property with the source instance. To compute the similarity between the two instances, a label-based approach is used if there are lexical similarities between several predicates, otherwise a structure based approach is used. Finally, the target instance with highest similarity is chosen for each source instance, and a final result is obtained by applying a threshold filter to the similarities of these matches. Because the candidate pair generation step makes a selective choice based on the existence of a common property, the overall size of the dataset to work with is reduced and thus a
lower runtime is obtained, but at the same time, this filtering may negatively affect the precision and especially the recall of the system.

2.6. STRIM

The operation of the STRIM [Khiat, Abderrahmane et al 2015] coreference resolution system is divided into three steps. First, it normalizes the input dataset using basic NLP techniques. During the normalization stage, the dataset passes through case conversion, lemmatization, and elimination of stop words. Afterwards, the edit distance string metric is applied to this normalized dataset to calculate the similarity between instance labels. Finally, each pair of instances with the most similar labels is selected as coreference. The STRIM system has yielded above 95% F-measure on both the sandbox and Mainbox coreference resolution tasks. It has a very straightforward approach with surprisingly good results.

2.7. SLINT++

SLINT+ [Nguyen, Khai, and Ryutaro Ichise 2013] is a schema-independent unsupervised learning approach to coreference resolution. In its predicate selection step, predicates with high frequency and diverse RDF objects are chosen and aligned between the two datasets. Only reliable alignments, whose confidence is greater than a threshold, will be kept for the next steps. A predicate is selected if its coverage, discriminability and harmonic means are greater than given threshold values. To accomplish the predicate alignment, predicates are first grouped based on their datatypes, namely, string, URI, integer, double and date, and then comparisons are
made. Similarity is then computed between two instances based on their shared values for these predicates.

2.8. SBUEI

SBUEI [TAHERI, A., & SHAMSFARD, M. 2013] iteratively applies similarity matching at both the instance level and the schema level in order to find matching instances between two ontologies. The system takes as input two equivalent concepts, called anchors, along with the ontologies to be aligned. The system starts by searching among the instances of the two anchor concepts to find instances with unique identity. Instances are compared by creating a Linked Instance Cloud (the instance, its neighbors, and neighbors of neighbors) for each instance. The overlap among the cloud for two instances determines their similarity. Individual instances in the cloud are compared based on their datatype properties only (object properties are ignored), using an edit distance string metric. Once one pair of matching instances is found, the algorithm checks for other matching pairs in the vicinity of this one. After all finding pairs of matching instances possible during this round, these are leveraged to find matching classes. Classes are compared by the degree of overlap among their instances. The instance matcher gives its feedback to schema matcher. This in turn is used to find two equal concepts. This result is again fed to instance matcher. This cyclic process continues until there are no instances or concepts left or until it is not possible to generate any more matches. SBUEI yields high values for precision, recall and F-measure. It performs very well even in the presence of data value transformations and structural transformations.
2.9. AgreementMaker

AgreementMaker [CRUZ, I et al. 2011] is an ontology matching framework that can work with a couple of algorithms to adjust to various alignment tasks. AgreementMaker runs a variety of matching algorithms on the input dataset and then fuses these results into a single result. AgreementMaker first tries to find candidate instances in the target ontology for each instance in the source ontology by using the labels of the instances and the type, if it exists. This step effectively reduces the number of comparisons. After preparing the list of candidate instances, the list is ranked based on the similarity between the source instance and each of the candidate instances. The system uses the string similarity method to compare labels, the vector space model approach to compare comments and literals, and property-value pairs to compare RDF statements. The system yields nice results, and the F-measures are just above 0.8. If a shared property is used across datasets, this approach can achieve even better results.

2.10. CODI

CODI [HUBER, J. et al. 2011], i.e. Combinatorial Optimization for Data Integration, is a probabilistic logical alignment system. The system derives from Markov logic and transforms the alignment problem into the maximum a posteriori optimization problem. In the first step, the identifiers, labels and annotations of individuals are preprocessed to remove stop words, camelCase strings are split into two words, and special characters are purged. The system then takes a small subset of all individuals, compares them with each other, and finds the lexical similarity. If this
similarity is more than a given threshold value, these pairs of individuals are added to the alignment. To compute this similarity, various string similarity metrics such as Cosine, Levenshtein, and Jaro Winkler are used, and an overall similarity value is obtained by taking either the average, maximum or weighted average of these metrics. Next, each match within the alignment is reconsidered by checking any additional individuals that are related to one of the individuals in the alignment through an object property. If the lexical similarity between any of these instances is greater than that between the related instance and the one it is currently matched with, this new instance replaces that one in the alignment. Thus, the anchor alignment is refined successively. To eliminate inconsistencies in the alignment, a coherence check is employed and inconsistent pairs are omitted for further processing. Finally, a greedy algorithm is used to ensure that the final alignment is one-to-one.

2.11. Serimi

Serimi [ARAUJO, S et al 2011] consists of two phases, namely, a selection phase in which a set of candidate instances is generated by comparing instance labels in each of the datasets, and a disambiguation phase where items found in first stage are filtered. The selection phase collects all of the literal values in the range of datatype properties related to a particular individual in the source ontologies. This collection of values is considered that individual’s “label”, and it is used to find potential matches in the target ontology by finding those that have the same or similar labels using SPARQL queries. This is called the pseudo homonym set. This set is filtered in a by using the RSWA string similarity algorithm to compare labels and
only keeping those with a similarity higher than a threshold. Finally, in a disambiguation phase, the individual within the candidate pseudo homonym set that has the highest similarity to the source individual is chosen as the coreference. In this phase similarity is computed using a model called RDS – Resource Description Similarity. This can be computed even when there is no direct ontology alignment between the source and target datasets.

2.12. Zhishi.links

Zhishi.links [NIU, X et al 2011] is built upon the concept of a distributed framework to index and process semantic resources using a scalable matching mechanism. As one-to-one matching consumes tremendous resources, Zhishi.links employs the concept of indexing the concepts before finding similarity between candidates. If an individual has aliases, they are also used in the indexing process. Individuals that are indexed to the same value are then considered as potential matches and a more in-depth semantic similarity metric is computed. Here if two individuals have a property – value pair in common, they are considered to have more semantic resemblance, and their similarity measure is increased. During a final stage, candidates are sorted based on their total similarity score to find the matching pair.

2.13. ASMOV

ASMOV [JEAN-MARY, Y. R., & KABUKA, M. R. 2011] is a general ontology alignment system rather than specific to coreference resolution. It uses a weighted average of similarity metrics along four different features of ontologies, obtains a pre-alignment based on these measurements, and then semantically verifies
this alignment to ensure that it does not contain semantic inconsistencies. The four features considered are (1) lexical elements like id, label and comments, (2) relational structure, (3) internal structure like type, domain and range of properties, and (4) extension i.e. instances of classes and property values. To begin with lexical similarity is calculated for each pair of concepts, properties and individuals. The system can use either UML meta thesaurus or WordNet to find a lexical similarity measure, otherwise a text matching algorithm can be used. Using the lexically similar entities as anchors, similarities for relational structure, internal structure and extensional dimension of nearby entities are calculated. This is used to find the overall similarity measure between entities by computing the weighted mean of the four individual similarity measures. These values are then filtered based on a threshold, and semantic consistency checking is used to remove inconsistent matches from the final alignment. Specifically for individuals, they will be considered more similar if their labels are similar, if they have the same type, or if they have the same properties.

2.14. LN2R

LN2R [SAÏS, F. et al 2010] considers data alignment as two reconciliation problems. The first is schema reconciliation i.e. mapping between concepts and relations, and the second is data reconciliation i.e. deciding if different descriptions refer to the same individuals e.g. the same person, same article or same gene. The second of these is relevant to coreference resolution. The system uses two approaches, namely informed and global. The informed approach uses knowledge
directly declared in the ontologies, together with string similarity metrics or information from a source such as WordNet, to compare two individuals. The global approach uses reasoning over “reference reconciliation” facts to infer whether or not two individuals are the same. Once two entities are determined to be the same, a fact to that effect is asserted. For example, if two people are known to be the same, then the organizations those people work at may be asserted to be the same as well. Horn Rules are then used to logically reconcile these facts, which may involve throwing out some matches and adding others. These logical reasoning results are enhanced by a numerical method for reference reconciliation called N2R. Similarity is modeled in the form of an equation that treats the similarity between two individuals as variables with unknown values, and the similarities between attributes as constants that are typically obtained by using the WordNet thesaurus. The algorithm exhibits good precision on the OAEI instance matching datasets.

2.15. ObjectCoref

ObjectCoref [HU, W. et al 2010] system is based on the concept of providing self-training to induce semi-supervised learning to resolve object coreference on the semantic web. The self-training is based on the formal and explicit semantics of owl:sameAs, owl:InverseFunctionalProperty, owl:FunctionalProperty etc. For example, owl:sameAs indicates that all URIs linked through this property must have the same identity. And if a property is an owl:InverseFunctionalProperty, then for each use of the property, the object uniquely determines the subject. Coreferent URIs are used to learn discriminability, and then they are further used to form a training set.
This training set allows the system to learn rules for comparing individuals based on their property/value pairs. Note that these rules can be based on any properties (and their values), not just the ones used to form the training set. The resulting rules are then used to assess the similarity between all pairs of individuals in the source and target ontologies, and those that are greater than a threshold value are included in the final alignment. The utility of this approach depends on the presence of sameAs, inverseFunctionalProperty, etc. in the ontologies to be aligned.

### 2.16. Anchor-Flood

Anchor-Flood [SEDDIQUI, M. H., & AONO, M. 2009] creates a “semantic linked cloud” for each individual and assesses the similarity between two individuals by measuring the similarity between their clouds. The clouds consist of the concepts linked to the individuals as well as their property-value pairs. Once two individuals are matched, other individuals with the same type are compared. Note that these individuals will now have a higher similarity, since at minimum their types will now match in their semantic linked clouds. This iterative process continues until no new matches are added.

### 2.17. Analysis of Current Approaches

An analysis of all of these coreference resolution systems shows that they can be classified into several general approaches.
2.17.1. Direct comparison of all pairs of individuals

These simple approaches often perform surprisingly well but are not scalable to large datasets. Examples include STRIM, SLINT++, and ObjectCoref. Systems in this category sometimes do some analysis prior to beginning the matching task to determine what information to use to compare two individuals. For example, SLINT++ attempts to choose properties with good coverage and strong discriminating power while ObjectCoref uses sameAs and inverseFunctional properties to create a training set that allows it to learn weightings for property-value pairs.

2.17.2. Comparison of individuals based on a “cloud”

In these approaches, a collection of values is created for each individual, and two individuals are compared based on some set-similarity metric over their collections. The things that are included in the collections vary from system to system. For example, InsMT+ includes the individual’s label and the names of any properties specified for that individual. SBUEI includes the individual’s label, its datatype property values, and the datatype values of its direct neighbors. Anchor-Flood is similar to SBUEI except that it includes the property names in addition to their values, and only for the individual in question rather than its neighbors.

2.17.3. Two phase comparison: coarse- and fine-grained

These algorithms avoid doing expensive comparisons between each individual in the source ontology and every individual in the target ontology by using faster but less accurate comparisons to find candidate match pairs that are then compared using
more expensive techniques. Some approaches for the coarse-grained comparisons are finding all target individuals that have any data property value exactly in common with the source individual (EXONA and Serimi), finding all individuals that share a common property (RiMOM), and finding all individuals of the same type with a somewhat similar label (AgreementMaker) or an exact match on label or alias (Zhishi.links). The fine-grained comparisons then try to find the best match from among all of the candidates based on either label (EXONA, RiMOM, AgreementMaker, and Serimi) or property names and values (Zhishi.links).

2.17.4. Reformulation as a different type of problem

Several systems reformulate the coreference resolution problem as a different type of problem. For example, Lily creates a subgraph for each individual based on an electrical circuit model, CODI transforms the alignment problem into the maximum a posteriori optimization problem using Markov logic, and LN2R treats establishing similarities between individuals as a system of equations.

2.18. Comparison Methodology Used By Current Systems

Table 1 summarizes what information each system uses to compare two individuals. In some cases the table cell contains ?, which indicates that there was not enough information available to determine the corresponding value for that system. The comparison metric column lists the most specific information that can be gathered from the literature.

Most systems use an individual’s label when comparing it to another individual, while the use of property names, property values, or both in the
comparison is less uniform. Among these systems, only SLINT+ compares property values differently based on their datatype. That system compares strings and URIs using TF-IDF cosine similarity, numbers using inverted disparity between the values, and dates using exact literal matching.

<table>
<thead>
<tr>
<th>Name of System</th>
<th>Individual’s Label</th>
<th>Property Name</th>
<th>Property Value</th>
<th>Comparison Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXONA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Edit Distance</td>
</tr>
<tr>
<td>InstMT+</td>
<td>X</td>
<td>X</td>
<td></td>
<td>Levenshtein, Jaro, and SLIM-Walker</td>
</tr>
<tr>
<td>Lily</td>
<td>X</td>
<td>?</td>
<td>?</td>
<td>String Similarity</td>
</tr>
<tr>
<td>LogMap</td>
<td>X</td>
<td>?</td>
<td></td>
<td>String Similarity</td>
</tr>
<tr>
<td>RiMOM</td>
<td>X</td>
<td>X</td>
<td></td>
<td>String Similarity</td>
</tr>
<tr>
<td>STRIM</td>
<td>X</td>
<td></td>
<td></td>
<td>Edit Distance</td>
</tr>
<tr>
<td>SLINT+</td>
<td></td>
<td>X</td>
<td></td>
<td>Datatype-specific Metrics</td>
</tr>
<tr>
<td>SBUEI</td>
<td>X</td>
<td>X</td>
<td></td>
<td>Edit Distance</td>
</tr>
<tr>
<td>AgreementMaker</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>String Similarity</td>
</tr>
<tr>
<td>CODI</td>
<td>X</td>
<td>(Object values)</td>
<td></td>
<td>Cosine, Levenshtein, and Jaro Winkler</td>
</tr>
<tr>
<td>Serimi</td>
<td>X</td>
<td>X</td>
<td></td>
<td>RWSA</td>
</tr>
<tr>
<td>Zhishi.links</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>ASMOV</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>WordNet or String Similarity</td>
</tr>
<tr>
<td>LN2R</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>WordNet or String Similarity</td>
</tr>
<tr>
<td>ObjectCoref</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>Anchor-Flood</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>?</td>
</tr>
</tbody>
</table>

*Table 1: Comparison Methodology of Current Coreference Resolution Systems*
Chapter 3

3. Methods

As mentioned in the introductory chapter, the primary goal of this work is to develop an analysis framework that is capable of evaluating the effect of various design decisions on a coreference resolution task. The design decisions to be considered are (a) What information is used to compare two individuals (property name, property value, or both), (b) How this information is numerically evaluated (comparing all values as strings, or using datatype-specific comparisons) and (c) whether or not the results of this numerical evaluation should be considered a match (by varying the threshold value). As is evident from the review of current coreference resolution systems presented in chapter 2, many systems made different choices regarding these design decisions. Therefore an analysis framework that can provide greater insight into their impact will be of considerable use to this research field. This chapter presents the capabilities of the analysis framework, the system architecture, and a step-by-step description of its use. More detailed information about the system in the form of UML diagrams can be found in Appendix A.

3.1. Capabilities

This section describes the capabilities of the coreference resolution system performance analysis framework.
3.1.1. Supported Input and Output Formats

The system is used to obtain coreference between two ontologies. The user can choose the files containing the ontologies to be matched through a file open dialog. Various ontology file formats are supported. Additionally, the user can specify a Reference Alignment File that contains the correct coreference for the chosen ontologies. With this information, the system will perform a true/false analysis after the coreference algorithm is run, compute common performance metrics such as number of matches, true positives, false positives, false negatives, precision, recall, and F-measure, and display the results on a graph. The results are also stored in a csv file that can be imported into a spreadsheet program for further analysis.

3.1.2. Configurable Comparison Criteria

The system allows the user to select what information will be used to compare two individuals: the individuals’ labels (referred to as instance parameter in the system), the property names (referred to as instance names in the system) or the property values (referred to as instance values in the system). Combinations of the above are also possible, e.g. both property names and values but not labels. This allows the framework to analyze differences in performance based on design decision (a) above.

3.1.3. Configurable Similarity Metrics

The system allows the user to choose what similarity metric to use for string-based comparisons. The similarity metrics currently implemented are Levenshtein, Jaro Winkler, and Longest Common Substring.
The user can opt to compare all values as strings (referred to in the system as a non-parametric comparison) or to compare values differently based on their data type (called a parameterized comparison). In the second case, strings are still compared using the chosen string similarity metric (Levenstein, Jaro Winkler or LCS) while numbers are compared based on their numeric difference and dates are compared based on either their year, year and month, or year, month, and day, as configured by the user. This gives more accurate results in cases in which two values have a big lexical difference and yet are actually very similar, such as 999,999 and 1,000,000 or 31 Dec 2015 and 1 Jan 2016. Together, these capabilities allow the framework to analyze the impact of design decision (b) above.

3.1.4. Configurable Thresholds and Automatic Variation

The use can specify the threshold for considering two labels, property names, and property values separately. Additionally, it is possible to fix the threshold for some things and vary it systematically for others, in order to quickly evaluate the sensitivity of the results to a particular threshold. For example, the user could set the similarity threshold for individual labels and parameter values to the constants 80% and 95%, respectively, and then have the framework systematically vary the threshold for property names from 50% to 100% in increments of 10%. The impact of this on performance is then automatically graphed. This functionality allows the framework to analyze the impact of design decision (c) above. Additionally, it can be useful for coreference resolution practitioners. As discussed in chapter 2, many coreference resolution systems involve setting threshold values. Unfortunately, there is little guidance on how to determine an appropriate value. This is generally done
empirically for a representative dataset. The capability of this analysis framework to automatically vary a threshold and evaluate the impact makes the process of arriving at an appropriate threshold value faster and easier.

3.2. Architectural Overview

The following figure illustrates the architectural diagram of the coreference resolution performance analysis framework. Various segments of the system are explained below.

![Architectural Overview of the System](image)

3.2.1. Input

As mentioned previously, the system can read in the two input ontologies in various formats, including owl, rdf, and ttl. The system also needs to read a Reference Alignment File to perform a truth analysis on the results. The expected format for the reference alignment file is a text file that contains a pair of URIs representing the matching individuals on each line, separated by a delimiter.
One thing that was an issue throughout this project was efficiency concerns. There are existing libraries to do some of the things required for this project, but these libraries are sometimes not efficient and run slowly for large files. Because of this, the OWL API was used to read in ontologies from the various file formats, but then the information relevant to the individuals was stored in an ArrayList. The pseudo code for this is shown below:

```java
getOntologyList

input : OWLOntology ont
output : List <InstanceData>
create Set entity of type OWLNamedIndividual
entity ∪ ont
for each individual in entity
    create Set sig ← individual using AnnotationAssertionAxiom
    for each OWLAnnotationAssertionAxiom axiom in sig
        Extract InstanceParameter, InstanceName, InstanceValue, DataType and URI
        Create an object of type InstanceData
        Add to List <InstanceData>
return List<InstanceData>
```

Algorithm 1: Get Ontology List

A similar issue occurred for the Levenshtein string similarity metric. The SecondString Java library has implementations for many different string similarity metrics, but the one for Levenshtein is quite slow. It takes nearly five times as long to run as the Jaro Winkler method, even though it is somewhat similar. Because of this, the Levenshtein algorithm has been implemented directly in this project. The pseudo code for this is shown below.
algorithm **LevenshteinSimilarity**

input: String s1, String s2
output: double stringSimilarityMeasure

L1 = length of s1
L2 = length of s2
longerLength ← Longer(L1, L2)
shorterLength ← Shorter(L1, L2)
return (longerLength – editDistance(longer string, shorter string) / longerLength

editDistance(Longer string s1, shorter string s2)
for count of each character i in s1
    integer lastvalue = i
    for count of each character j in s2
        if i == 0
            cost [j] = 0
        else
            if j ≠ 0
                \[d_{i,j} = \text{MIN} \left\{ \begin{array}{l} d_{i-1,j} + C_{del}(b_i) \\ d_{i-1,j} + C_{ins}(a_j) \\ d_{i-1,j-1} + [a_i ≠ b_i] \times C_{sub}(a_j, b_i) \end{array} \right.\]

Where **C_{operation}** represents the fixed cost of operation as under:

**C_{del}** represents cost of delete operation = 1. After delete operation move-down vertically in the matrix.

**C_{ins}** represents cost of insertion operation = 1. After insertion opration move horizontally along the row in the matrix.

**C_{sub}** represents cost of substitution operation = 1. After substitution operation move diagonally down in the matrix.

*Algorithm 2: Levenshtein Similarity*

---

System

The main system is comprised of three modules, namely Comparison, Data Collection, and Evaluation. The Comparison module is used to compare the individuals within two ontologies based on their labels, property names, and property values at the thresholds specified by the user. The comparison is based on an algorithm chosen by the user. The Data Collection module varies the threshold for one of the above-mentioned parameters while keeping the other two constant, in order to analyze the effect of the threshold value on the results. The Evaluation module uses the output obtained from the other modules to generate a truth analysis on the results and generate the output. The most important methods among these modules are the ones that perform the comparison of two individuals. These methods are therefore described in more detail.

The findMatchesNonParametrically method, which compares all values as strings, takes as input the list of individuals in each ontology, together with the data type properties they are involved in. Additionally, the method takes boolean values indicating whether or not the labels, property names, and property values should be used in the comparison. The method also takes double values indicating the thresholds for considering two labels, property names, and property values to be similar. Finally, the method takes a string indicating which string similarity metric to use for the comparisons, Levenstein, JaroWinkler, or LCS. The method then compares every individual from the first ontology to every individual in the second ontology. Each pair of individuals is compared based on the chosen criteria (labels,
property names, and/or property values), and the pair is added the coreference to the result set if the similarities are greater than their corresponding thresholds.

The findMatchesParametrically method, which does a datatype-specific comparison of two entities, is very similar to the previous method. One difference is that the threshold value for numeric datatypes is now interpreted as the maximum difference between the two values divided by their mean that will be considered a match. The other difference is that an additional parameter to the method indicates how the user has elected to compare two dates (based on some combination year, month, and day). These fields are then used to compare date values.

3.2.2. Output

As mentioned previously, the system is designed to generate output in three different forms. When a single configuration is run, the user is presented with the matches that are obtained in a tabular format within the user interface. If the user decides to run a configuration repeatedly, with automatic variation of one of the threshold values, the system creates comma-separated value output files for each of the threshold values that are considered and stores them on the computer. The system is also able to display the information from these files graphically within the interface. The information contained in these files and graphs is described below.

**True Positives**: These are the correct matches according to the reference alignment file that were also identified by the system.

**False Positives**: These are the matches that were identified by the system but which are not correct according to the reference alignment file.
**False Negatives:** These are the results that are correct according to the reference alignment file and ought to be a part of the result, but that are not recognized by the system as correct.

**Precision:** Precision can be defined as the ratio of the number of correct matches produced by the system to the total number of correct and incorrect matches it produced. In Figure 4 it is equivalent to $p = \frac{A}{A+C}$. Intuitively, precision represents how often the coreferences returned by the system were valid.

**Recall:** Recall can be defined as the ratio of the number of matches produced by the system to the total number of matches that exist in the reference alignment file. In Figure 4 it is $r = \frac{A}{A+B}$. In simple words, recall how many of the valid coreferences that system was able to find.

![Figure 4: Concept of Precision and Recall](image)

The analysis framework is able to graph each of these quantities, and to put multiple quantities on the same graph for comparison. Figure 5 provides an example. This data
was collected by comparing individuals based on their labels, property names, and property values. In this case all data was compared as strings using the Levenshtein algorithm. The thresholds for property names and values were fixed at .25 and .90, respectively. Meanwhile the threshold for the label similarity was varied and the impact on performance was measured. In the graph the red line indicates the number of true positives and the blue line indicates the number of false positives (the y-values). The x-axis shows the results of varying the threshold for the label similarity.

![Graph](image)

*Figure 5: Sample Output Graph*

### 3.3. System Operation

This section provides a step-by-step illustration of a typical process flow for obtaining coreference between any two given ontologies and analyzing their accuracy.

1) The user has to select two ontology files from the computer’s file system.
2) The user selects a string similarity metric to be used for making comparisons between string values (in the parametric case where datatype is considered) or between all values (in the non-parametric case where datatype is not considered).
3) The user selects a threshold value for comparing individual’s labels (called instance parameter in the system).

4) The user makes a choice either to make a plain comparison (everything as a string) or parameterized comparison (where datatype is relevant).

![Figure 9: Selecting Thresholds for Non-parametric Comparison](image)

5) The user selects a threshold value for comparing property names (instance name).

6) The user selects a threshold value for property values (instance value).

7) If the user has chosen to do a parameterized comparison s/he selects precise parameterized parameter values, particularly on what fields to compare two dates (year, month, day). The user can also set different threshold values for numeric and string similarity in this case.

![Figure 10: Selecting Various Accuracies for Parametric Comparison](image)
8) If the user wants to see the effect of this configuration, s/he now press the Process Ontologies button.

9) The user is presented with tabular output showing what matches were obtained. 

![Figure 11: Tabular Result of Matches Found](image)

10) If the user’s goal is to consider the effect of variation of the threshold parameters, the user chooses which threshold to analyze: the one for individual labels, parameter names, or parameter values. The user can also specify what information to use for comparing two individuals: labels, property names, and/or property values. 

![Figure 12: Available Options to Vary Various Parameters](image)

11) The output files for varying the selected threshold from 50% to 100% are generated and stored with a specific naming convention.

12) The user is prompted to select the Reference Alignment File.
13) The user is presented with True/False Analysis as a set of graphs, as described in section 3.2.3.
Chapter 4

4. Results and Analysis

This chapter shows the usefulness of the coreference resolution analysis framework presented in chapter 3 by using it to explore several research questions. More work needs to be done to solidify these results, in particular by running similar experiments on other datasets, but they already show the utility of the framework for understanding the performance of design decisions made by typical coreference resolution systems.

The ontologies used for the analyses presented here are based on DBPedia and YAGO. Both of these datasets are related to Wikipedia – DBPedia is entirely from Wikipedia and YAGO is based on Wikipedia, WordNet, and GeoNames. Both of these linked datasets contain millions of individuals. The work presented here is based on a subset of 8685 individuals from DBPedia and 1680 individuals from YAGO. The correct coreference for many individuals can be determined because they point to the same page on Wikipedia. This information was used for the reference alignment.

4.1. Impact of the Information Used For Comparison

It is already known that entity labels are very useful when aligning two datasets [Cheatham, 2014]. In this section the utility of using property names and property values when comparing two individuals is explored using the coreference resolution
analysis framework. For this experiment the Levenshtein string similarity metric was used in all cases. In the table below, the left column of graphs shows the precision while the right side shows the recall. In the first row, only the parameter names were used to compare two individuals. In the second row, only the parameter values were used. In the third row, the comparison was based on both the names and values of the parameters associated with each individual. In some cases, the line is absent for higher threshold values. This is because no coreference were produced at this threshold level.

From these graphs we can see that recall is approximately the same regardless of whether parameter names, parameter values, or both are used to compare individuals. However, the precision is affected by this choice. In particular, precision is higher about 8 percent higher when parameter values are used than when parameter names are used. Using both names and values gives a further 2 percent increase in precision, without an impact on recall. This type of data can be very useful for the designers of the coreference resolution systems discussed in chapter 2.
4.2. Impact of the String Similarity Metric

The graphs below display the number of true positives (red line) and false positives (blue line) generated by the coreference resolution analysis framework when it is configured to use individual labels, property names, and property values for comparison. The thresholds for property names and property values are being held constant at 25% and 90%, respectively, and the threshold for individual labels is being varied in increments of 10%. All values are being compared as strings. The only difference is the string similarity metric used to do the comparisons.

Though more analysis is required, from these results we can note that the Longest Common Substring metric has a higher ratio of true positives to false positives than the other two metrics. This can be useful if a coreference resolution system has already gathered a set of candidate matches and is trying to filter them. Another thing we can note is that the optimal threshold value is different for different metrics.
Figure 14 Impact of the Levenshtein metric

Figure 15 Impact of the JaroWinkler metric

Figure 16 Impact of the LCS metric
5. Conclusions and Future Work

This chapter summarizes the results presented in this thesis and discusses future work related to this topic.

5.1. Conclusions

Many interesting observations and striking facts can be revealed if diverse data sources from the web can be accessed, queried and analyzed by both humans and machines in a coherent way, and this essentially requires using Linked Data. But different linked datasets are generally created independently from each other, are often unaware of the existence of each other, and hence are disconnected. Hence the need arises to link individuals in these different non-connected datasets. The attempt to establish sameAs relationships between two individuals from such different datasets is called coreference resolution.

As depicted in chapter 2, there exist many coreference resolution systems that try to perform the task of coreference resolution over diverse and disconnected datasets. To achieve their goal, these coreference resolution systems need to compare the instances from both datasets in order to determine whether or not they are a match. Each coreference resolution system makes a different set of design decisions related to how to do this. These design decisions typically include aspects like what combination of instance attributes like instance label, property label and property value should be
compared, the choice of algorithm to compare them, the choice of threshold, and the mode of comparison i.e. comparing all values as strings or taking into consideration the datatype of property values when doing the comparisons. Because these design decisions have a tremendous effect on the performance of a coreference resolution system, it is advantageous to have a framework capable of analyzing their impact on various performance metrics in a very detailed manner. This has been the major contribution of this thesis.

The framework can read the input dataset in the form of .rdf, .owl or .nt files. Currently three string similarity metrics are available to make comparisons, namely, Levenshtein, Jaro Winkler and Longest Common Substring. Comparisons between two individuals can be made on any combination of instance label, parameter name, and parameter value, and different thresholds can be specified for each of these. Additionally, it is possible to compare numeric and date values based on their datatype rather than as strings. The system computes a variety of common performance metrics, including true positives, true negatives, false positives, precision, and recall. Output is presented both tabularly and graphically and also stored as comma separated value files that can be exported into a spreadsheet program for further analysis. Additionally, the system can automatically vary a threshold value and graph the effect on accuracy, which allows the user to study the sensitivity of the approach to threshold value and also to select an appropriate threshold value.

Chapter 4 shows the utility of the analysis framework by using it to establish the impact of two design decisions on the accuracy of the coreferences that are generated.
In the first case, it was found that using parameter values rather than parameter names to compare two instances results in higher precision and roughly the same recall. Comparing using both parameter names and values improves precision very slightly over using parameter values alone. Secondly, it was found that the Longest Common Substring metric has a higher ratio of true positives to false positives than the other two metrics, which can be useful when filtering candidate match pairs. Both of these results are preliminary and need to be confirmed by further experimentation on different datasets.

5.2. Future Work

Several additions to the coreference system analysis framework are planned. In particular, support for additional similarity metrics and the ability to choose a different metric to compare labels, parameter names, and parameter values are intended in the near future, as is the conversion of the framework into a web application for easier access and use by other researchers who are studying coreference resolution. Additionally, the system can be modified to work with a linked data SPARQL endpoint rather than requiring ontology files.

Another area of future work is to use the analysis framework to solidify the results presented in chapter 4 by performing the same analysis on different datasets. Possibilities include the OAEI instance matching benchmarks and the GeoLink dataset. Both of these have at least a partial reference alignment that can be used to assess coreference performance. In addition, other research questions can be
addressed through the analysis framework, including the impact of datatype-specific comparison of property values.
6. References


DAMAK, SYRINE, HAZEM SOUID, MAROUEN KACHROUDI, and SAMI ZGHAL. "EXONA Results for OAEI 2015."


Appendix A: UML Diagrams

This appendix contains UML diagrams for each class in the application. The following symbols are used:

+ public visibility
- private visibility
# protected visibility
@ has getters / setters
Σ static method

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>OntologyMatcher()</td>
</tr>
<tr>
<td>initialize()</td>
</tr>
<tr>
<td>actionPerformed (ActionEvent)</td>
</tr>
<tr>
<td>itemStateChanged (ItemEvent)</td>
</tr>
<tr>
<td>StateChanged (ChangeEvent)</td>
</tr>
<tr>
<td>loadTable(List&lt;MatchedInstanceData&gt;)</td>
</tr>
</tbody>
</table>

Table 2: UML for the OntologyMatcher Class
### Ontologies

<table>
<thead>
<tr>
<th>iri</th>
<th>IRI</th>
</tr>
</thead>
<tbody>
<tr>
<td>df</td>
<td>OWLDataFactory</td>
</tr>
<tr>
<td>mngr</td>
<td>OWLOntologyManager</td>
</tr>
<tr>
<td>ont</td>
<td>OWLOntology</td>
</tr>
</tbody>
</table>

| processFile (File) | List<InstanceData> |
| readFile (File)    | List<InstanceData> |
| readDotNTFile (File) | List<InstanceData> |
| getOntologyList (OWLOntology) | List<InstanceData> |

| findMatchesNonParametric (List<InstanceData>, List<InstanceData>, double, double, double, URIAlignment, String) |  |

*Table 3: UML for the Ontologies Class*

### MatchedInstanceData

<table>
<thead>
<tr>
<th>-, @</th>
<th>instanceA</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>-, @</td>
<td>instanceB</td>
<td>String</td>
</tr>
<tr>
<td>-, @</td>
<td>parameterNameA</td>
<td>String</td>
</tr>
<tr>
<td>-, @</td>
<td>parameterNameB</td>
<td>String</td>
</tr>
<tr>
<td>-, @</td>
<td>parameterValueA</td>
<td>String</td>
</tr>
<tr>
<td>-, @</td>
<td>parameterValueB</td>
<td>String</td>
</tr>
<tr>
<td>-, @</td>
<td>dtvpA</td>
<td>String</td>
</tr>
<tr>
<td>-, @</td>
<td>dtvpB</td>
<td>String</td>
</tr>
<tr>
<td>-, @</td>
<td>uriValueA</td>
<td>String</td>
</tr>
<tr>
<td>-, @</td>
<td>uriValueB</td>
<td>String</td>
</tr>
</tbody>
</table>

| + | MatchedInstanceData (String, String, String, String, String, String, String, String, String, String) |
| + | int hashCode() |
| + | boolean equals() |
| + | String toString() |

*Table 4: UML for the MatchedInstanceData Class*
<table>
<thead>
<tr>
<th>LCS (Longest Common Substring)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- getLongestCommonSubstring(String, String)</td>
</tr>
<tr>
<td>- getLengthOfLCS (String, String)</td>
</tr>
<tr>
<td>+,Σ longestCommonSubstringSimilarity</td>
</tr>
</tbody>
</table>

*Table 5: UML for the LCS (Longest Common Substring) Class*

<table>
<thead>
<tr>
<th>JaroWinkler</th>
</tr>
</thead>
<tbody>
<tr>
<td>- strOne</td>
</tr>
<tr>
<td>- strTwo</td>
</tr>
<tr>
<td>- firstMatch</td>
</tr>
<tr>
<td>- secondMatch</td>
</tr>
<tr>
<td>- distance</td>
</tr>
<tr>
<td>- getMatch()</td>
</tr>
<tr>
<td>- getUnMatch(String, String)</td>
</tr>
<tr>
<td>- getCommonPrefix (String, String)</td>
</tr>
<tr>
<td>+ getSimilarity(String, String)</td>
</tr>
</tbody>
</table>

*Table 6: UML for the Jaro Winkler Class*

<table>
<thead>
<tr>
<th>IterativeMatching</th>
</tr>
</thead>
<tbody>
<tr>
<td>+,Σ findMatchesIteratively(List&lt;InstanceData&gt;, List&lt;InstanceData&gt;, double, double, double, boolean, String)</td>
</tr>
<tr>
<td>+,Σ writeMatchFile(String, String, List&lt;InstanceData&gt;)</td>
</tr>
<tr>
<td>+,Σ writeTFFile(String, double, double, double, String, boolean, String)</td>
</tr>
<tr>
<td>+,Σ readMappingFile()</td>
</tr>
<tr>
<td>+,Σ countSlashes(String)</td>
</tr>
</tbody>
</table>

*Table 7: UML for the IterativeMatching Class*
### InstanceData

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>+</td>
<td>InstanceData(String, String, String, String)</td>
<td>Constructor</td>
</tr>
<tr>
<td>+</td>
<td></td>
<td>int</td>
</tr>
<tr>
<td>+</td>
<td>equals(Object)</td>
<td>boolean</td>
</tr>
<tr>
<td>+</td>
<td>toString()</td>
<td>String</td>
</tr>
</tbody>
</table>

*Table 8: UML for the InstanceData Class*

### BasicUtilities

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>findMax(int[])</td>
<td>int</td>
</tr>
<tr>
<td>+</td>
<td>findMaxDouble(double[])</td>
<td>double</td>
</tr>
<tr>
<td>+</td>
<td>findMin(int[])</td>
<td>int</td>
</tr>
<tr>
<td>+</td>
<td>findMinDouble(double[])</td>
<td>double</td>
</tr>
<tr>
<td>+</td>
<td>isNumeric(String)</td>
<td>boolean</td>
</tr>
<tr>
<td>+</td>
<td>isDate(String)</td>
<td>boolean</td>
</tr>
<tr>
<td>+</td>
<td>getFormatOfDateString(String)</td>
<td>String</td>
</tr>
<tr>
<td>+</td>
<td>areDatesMatching(String, String, int)</td>
<td>boolean</td>
</tr>
</tbody>
</table>

*Table 9: UML for the BasicUtilities Class*