Lehigh University
Lehigh Preserve

Theses and Dissertations

2014

Towards a Linked Semantic Web: Precisely, Comprehensively and Scalably Linking Heterogeneous Data in the Semantic Web

Dezhao Song
Lehigh University

Follow this and additional works at: http://preserve.lehigh.edu/etd

 Part of the Computer Sciences Commons

Recommended Citation
Song, Dezhao, "Towards a Linked Semantic Web: Precisely, Comprehensively and Scalably Linking Heterogeneous Data in the Semantic Web" (2014). *Theses and Dissertations.* Paper 1634.

This Dissertation is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

Towards a Linked Semantic Web:

Precisely, Comprehensively and Scalably Linking

Heterogeneous Data in the Semantic Web

by

Dezhao Song

A Dissertation

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Computer Science

Lehigh University

January 2014

@ Copyright by Dezhao Song

All Rights Reserved

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Accepted Date

Committee Members:

Jeff Heflin, Committee Chair and Dissertation Advisor

Associate Professor, Lehigh University

Xiaolei Huang, Committee Member

Associate Professor, Lehigh University

Héctor Muñoz-Avila, Committee Member

Associate Professor, Lehigh University

Christopher G. Chute, Committee Member

Professor, Mayo Clinic

Acknowledgement

I would like to take this opportunity to express my sincere thanks to those who helped me in one way or another on conducting research and the writing of this dissertation.

I want to express my heartfelt thanks to my advisor, Professor Jeff Heflin. The work presented in this dissertation is not possible without his support and advice. I also want to thank Professors Xiaolei Huang, Hector Munoz-Avila and Christopher G. Chute for participating on my Ph.D. committee and for their insightful feedback, comments and suggestions. I would like to especially thank Professor Christopher G. Chute from Mayo Clinic for flying in to Lehigh to participate my final Ph.D. defense. I also want to sincerely thank Professor Xiaolei Huang for providing me the opportunity to participate in the cervical cancer project, which enabled me to get to know a brand new field and learn a lot of knowledge and skills.

I would like to show my sincere thanks to my father Zhaoxin Song and my mother Xinmin Pang for their endless support during my 5-year stay at Lehigh University. Starting from the beginning of my study at Lehigh, they kept encouraging and supporting me whenever I felt frustrated for my course work, my research and my life. Without their love and support during the years, I would not have come to where I am. I want to apologize for not calling back a lot when I had to spend a lot of time for my research but I want to take this opportunity to express my thanks and love for my parents. Thank you so much for always being there for me.

I gratefully acknowledge the help from many members who are present and former of the Semantic Web and Agent Technologies (SWAT) Lab at Lehigh University. I thank Xiangjian Zhang and Sambhawa Priya for collaborating on the Tag Could work, Xingjian Zhang, Yang Yu, Sambhawa Priya and Yi Luo for working together on the ADEN project, and Xingjian Zhang, Yingjie Li and Yang Yu for their insightful suggestions and comments

on the dissertation and my daily research.

While I was working on my dissertation, I was also research intern at Mayo Clinic working with Professor Cui Tao, Professor Christopher G. Chute, and Deepak Sharma, and at IBM Ireland Research Center working with Dr. Freddy Lecue and Dr. Anika Schumann. These internship opportunities provided me the platform for applying my Semantic Web and Entity Coreference knowledge to real world application scenarios. The experiences would be really helpful for my future career.

It would be extremely difficult for me (a non-native English speaker) to express my thanks for all the people that have provided your help and support to me during these years in English in the best way. All in all, thanks everyone for your help, support, and love.

Contents

List of Tables

List of Figures

Abstract

The amount of Semantic Web data is growing rapidly today. Individual users, academic institutions and businesses have already published and are continuing to publish their data in Semantic Web standards, such as RDF and OWL. Due to the decentralized nature of the Semantic Web, the same real world entity may be described in various data sources with different ontologies and assigned syntactically distinct identifiers. Furthermore, data published by each individual publisher may not be complete. This situation makes it difficult for end users to consume the available Semantic Web data effectively. In order to facilitate data utilization and consumption in the Semantic Web, without compromising the freedom of people to publish their data, one critical problem is to appropriately interlink such heterogeneous data. This interlinking process is sometimes referred to as ***Entity Coreference***, i.e., finding which identifiers refer to the same real world entity. In the Semantic Web, the ***owl:sameAs*** predicate is used to link two equivalent (coreferent) ontology instances. An important question is where these *owl:sameAs* links come from. Although manual interlinking is possible on small scales, when dealing with large-scale datasets (e.g., millions of ontology instances), automated linking becomes necessary.

This dissertation summarizes contributions to several aspects of entity coreference research in the Semantic Web. First of all, by developing the *EPWNG* algorithm, we advance the performance of the state-of-the-art by 1% to 4%. *EPWNG* finds coreferent ontology instances from different data sources by comparing every pair of instances and focuses on achieving high precision and recall by appropriately collecting and utilizing instance context information domain-independently. We further propose a sampling and utility function based context pruning technique, which provides a runtime speedup factor of 30 to 75. Furthermore, we develop an on-the-fly candidate selection algorithm, *P-EPWNG*, that enables

the coreference process to run 2 to 18 times faster than the state-of-the-art on up to 1 million instances while only making a small sacrifice in the coreference F1-scores. This is achieved by utilizing the matching histories of the instances to prune instance pairs that are not likely to be coreferent. We also propose *Offline*, another candidate selection algorithm, that not only provides similar runtime speedup to *P-EPWNG* but also helps to achieve higher candidate selection and coreference F1-scores due to its more accurate filtering of true negatives. Different from *P-EPWNG*, *Offline* pre-selects candidate pairs by only comparing their partial context information that is selected in an unsupervised, automatic and domain-independent manner.

In order to be able to handle really heterogeneous datasets, a mechanism for automatically determining predicate comparability is proposed. Combing this property matching approach with *EPWNG* and *Offline*, our system outperforms state-of-the-art algorithms on the 2012 Billion Triples Challenge dataset on up to 2 million instances for both coreference F1-score and runtime. An interesting project, where we apply the *EPWNG* algorithm for assisting cervical cancer screening, is discussed in detail. By applying our algorithm to a combination of different patient clinical test results and biographic information, we achieve higher accuracy compared to its ablations. We end this dissertation with the discussion of promising and challenging future work.

Chapter 1

Introduction

1.1 Motivation

The World Wide Web (WWW) has significantly changed the way people share and access knowledge by building a global information space. Hypertext links on web pages allow people to surf through this global information space using Web browsers. Search engines index the documents to enable people to efficiently look up the information they are interested in and are generally able to provide decent query results. This basic search functionality has been enabled by the generic, open and extensible nature of the Web, which is also seen as a key feature in the Web's unconstrained growth. Even further, search engines now are also able to analyze query logs in order to provide personalized results to end users. By witnessing the success of the World Wide Web, particularly search engines giants such as Google, Yahoo and Bing, people may start to believe that the Web has reached its full potential as a global knowledge and information repository. However, with the amount of data available on the Web rapidly increasing in recent years, some of people's needs cannot be really fulfilled by today's most advanced web technologies.

As a concrete example, suppose we are issuing the following query to the Web: "find all conferences that Tim Berners-Lee have published in". In this example, search engines will have difficulties in finding the answers because contemporary search technologies are primarily keyword-based. A document/web page is determined to be relevant to a user's query if its content is "similar" (enough) to the text (in the query) entered by users. The basic determinant of this similarity is the one between textual representation of the words in the user's query and the documents in the web. However, search engines are not able to understand the query correctly, particularly the relationships between those different query conditions or "things" in it. Although it sounds like humans should be able to understand the query and look up the answers from various different data sources, search engines are currently not able to do this in an automated manner. Furthermore, the data on the traditional web is represented as documents or free text, and there is no structure or relationships between the data. Even if the query can be "understood" by search engines so that the relationships between different conditions are parsed correctly, the data on the current web is not designed for answering such structured queries.

In order to help the Web to really reach its full potential, the Semantic Web [1] has suggested a way of extending the existing web with structure and providing a mechanism to specify formal semantics that are machine-readable and shareable. In this way, the structured information on the Semantic Web can be readily interpreted by machines, so they can perform more of the tedious work involved in finding, combining and acting upon information on the web without human intervention. This is accomplished by using ontologies. An ontology is a formal logic-based description of a vocabulary that allows one to talk about a domain of discourse. The vocabulary is articulated using definitions and relationships among the defined concepts. As ontologies use formal logic, they can describe a domain

unambiguously as long as the information is interpreted using the same logic. Moreover, the use of logic makes it possible to use software to "infer" implicit information in addition to what is explicitly stated.

With the development and recognition of the Semantic Web, in recent years, more and more people, academic institutions and even commercial companies have already started to publish their data in Semantic Web formats, such as the Resource Description Framework (RDF)[1] and the Web Ontology Language (OWL)[2]. However, even when data is represented in a structured way using Semantic Web formats, it still may not be sufficient for handling the query discussed above. First of all, the nature of the Semantic Web is that data is not hosted at a central place but rather stored in a distributed manner in many different places/data sources. And different data publishers may publish their data by adopting different ontologies or schemas. Moreover, one observation is that one real world entity (e.g., people, geographical locations, organizations, musics, books, etc.) may be described and published by many data publishers with syntactically distinct identifiers. For example, CiteSeer [2] and DBLP [3] are two major academic databases where people can look up academic publications in a variety of research fields. Each of the two databases identifies *Tim Berners-Lee*, the inventor of the World Wide Web and the visionary of the Semantic Web, with their own identifiers and describes him in distinct ways with complementary information. However, such identifiers from different data sources are not linked to each other and thus end users of such databases may have to query each individual data source to obtain relatively comprehensive information for the entity of interest. Consequently, this single conceptual query actually requires multiple physical queries to various data sources in order to obtain a relatively comprehensive set of answers.

[1] http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/
[2] http://www.w3.org/TR/owl-ref/

One of the most exciting things about the Semantic Web is to drive the evolution of the Web as a global information space from a Web of documents to a Web of data, where not only documents but data is also linked. Linked Data[3] [4] is one notable effort for people to publish, share and connect data in the Semantic Web. These data are often independently generated and distributedly stored in many locations, and are also heterogeneous, covering diverse domains such as academia, entertainment, arts, biology, government, geography, etc. In today's Semantic Web, there are many billions of semantic data triples publicly available in the above mentioned various domains. Linking Open Data (LOD)[4] is a project that aims to establish this huge, distributed, heterogeneous and connected data hub by publishing various open datasets as RDF on the Web and by setting RDF links between data items from different data sources.

According to the latest statistics[5], there are currently 295 datasets from various domains (e.g., People, Geographic, Publications, Media, etc.) in the LOD cloud with more than 31 billion triples and about 500 million links across different datasets. The links are outgoing links that connect each dataset to the others. One important type of such links is the *owl:sameAs* links that connect equivalent ontology instances in the Semantic Web. With the help of *owl:sameAs* links, a system will then be able to enter this interlinked Semantic Web repository and then retrieve all available information for the same entity by following such links. Take the above example again. As shown in Figure 1.1, without such *owl:sameAs* links, end users will have to issue separate queries to various data sources to obtain the information; while when *owl:sameAs* links do exist, a single query will then do the same.

[3]http://linkeddata.org/
[4]http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[5]http://www4.wiwiss.fu-berlin.de/lodcloud/state/

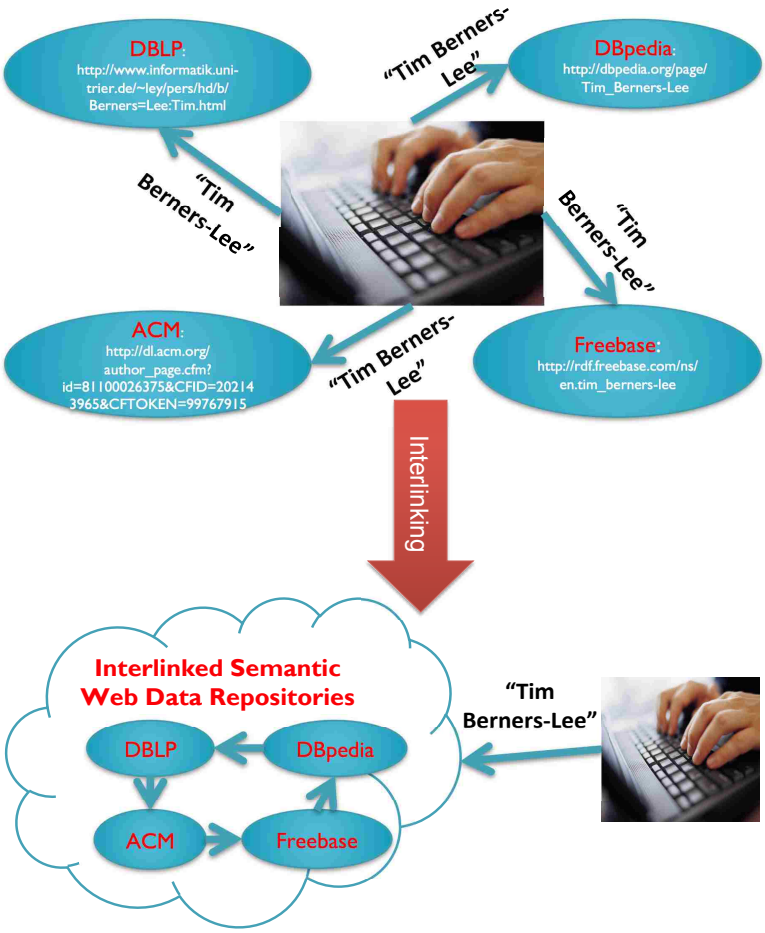


Figure 1.1: Data Browsing with Data Linkages

Talking from query perspective, such *owl:sameAs* links could also facilitate the query answering process. Figure 1.2 demonstrates a concrete example. Here, we have two person instances: *Person1* and *Person2* from two different data sources; each of them has some associated information respectively, such as name, affiliation, publication, friendship, etc. Suppose we issue the following query: "find people that are from MIT and also know Jim Hendler". Neither data source would be able to answer this query, since they only contain partial information and thus are not able to satisfy all conditions specified in the query. Now, if we set up an equivalence linkage between the two person instances, i.e., connecting them with the *owl:sameAs* predicate, things will become different. With such

a linkage here, we actually construct a new virtual instance that covers the information of both isolated instances, which enables the query constraints to be met. This example clearly demonstrates the great benefits of having *owl:sameAs* links between instances across different data sources.

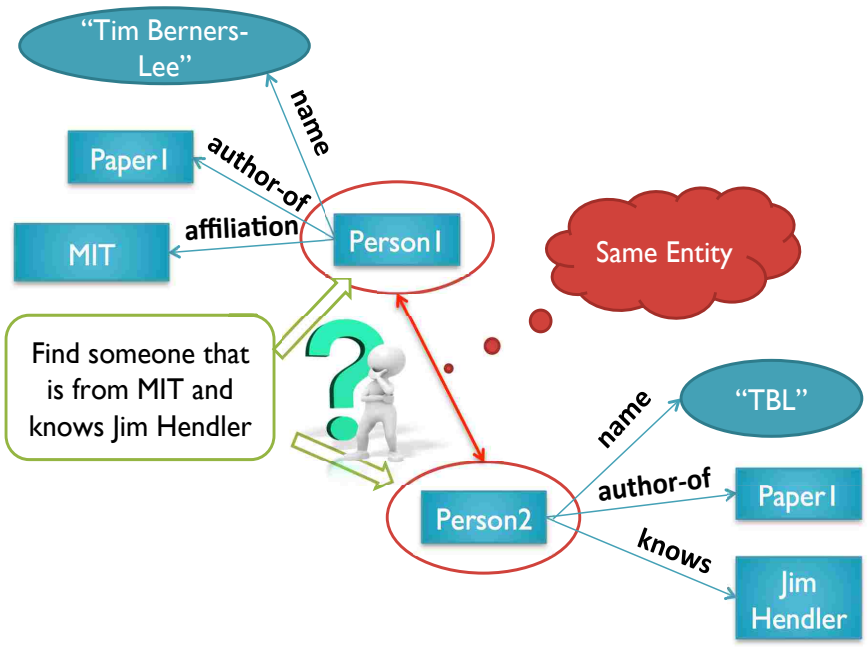


Figure 1.2: Facilitated Query Answering with Equivalence Relationship

The process of producing such *owl:sameAs* links, or interlinking, has been studied by Natural Language Processing researchers as the *Entity Coreference* [5] and *Entity Resolution* [6] problems and by Database researchers as the *Deduplication* [7] and *Record Linkage* [8] problems. The purpose of entity coreference is to decide the real world entity that a mention refers to. A mention could be an occurrence of a person name, a publication title or a geographical location name in a document, a web page, etc. For example, in different news articles, two or more mentions of the name *James Henderson* may exist and an entity coreference algorithm should answer which real word person entity that these two mentions

actually refer to. In the Semantic Web, we extend the concept of a mention to include Semantic Web instances. An instance is identified with a Uniform Resource Identifier (URI) while syntactically distinct URIs could actually represent the same real world entity. We will use the term *Entity Coreference* to refer to the process of finding ontology instances that refer to the same real world entity throughout this dissertation.

As recently reported by Halpin et al. [9], only 50% ($\pm$ 21%) of the *owl:sameAs* links in the LOD cloud are correct. Therefore, one big challenge for Linked Data and the Semantic Web is to scalably establish high-quality coreference relationships between ontology instances from different data sources, so that the datasets can be interlinked through such equivalences in order to facilitate upstream applications, such as information integration and query answering [10, 11]. Producing high quality equivalence relationships for the current Semantic Web is a non-trivial task. Although it might be possible for performing manual linking on small datasets [12], automatic approaches will be needed for detecting equivalence linkages across large-scale heterogeneous datasets.

Given this target of interlinking large-scale and heterogeneous Semantic Web data, we are facing several challenges. First of all, in order to detect coreferent instances precisely and comprehensively, it is important to locate and utilize the relevant information (the context) of the instances in an appropriate way. There are various situations that can mislead the entity coreference results. Name variations, the use of abbreviations, and misspellings can all play a role in the final results. Also, the collected data may come from heterogeneous data sources and may not be complete. For instance, for a given person instance, different aspects of this person may be described in different data sources. One source may provide the name and affiliation of this person while another source can have other types of information, such as name, date of birth, email address, etc. In addition, even though some information is

present, it may be noisy data. For example, some date information may be included in the context of this person and is treated as date of birth; however, such date information could simply be the date of a social event that this person attended. To ensure the quality of the generated links, an entity coreference algorithm needs to be able to address such challenges appropriately.

Furthermore, making this context selection and utilization process domain-independent is equally important. A domain refers to the category (e.g., People, Geographic, Publications, Media, etc.) and the usage (e.g., academic people, politics, etc.) of the data. In the past, domain-specific techniques have successfully helped to achieve good entity coreference results, e.g., relying on matching person names to identify coreferent person instances. However, when considering various domains, humans may lack the knowledge or time to specify what information to utilize and thus coreference tools are less likely to be available for all domains end users deal with.

Last but not least, scalability issues need to be taken into account when designing an entity coreference algorithm, considering the current scale (295 datasets and 31 billion triples) of the data in the Semantic Web. Much of the prior work [13, 5, 14, 15, 16] has adopted the simple approach of resolving the coreference relationships between instances by comparing every pair of instances. However, a single dataset from Linked Data could have millions of instances. For example, RKB[6] [17], a well-known Semantic Web knowledge base for academic publications, contains at least 3 million instances describing researchers. In order to scale to such a large volume of data, one possible solution would be to adopt a filtering process, i.e., selecting instance pairs that are likely to be coreferent through a lightweight manner and then only applying the actual and expensive entity coreference

[6]http://www.rkbexplorer.com/data/

algorithms on such similar pairs. Another possible direction to scaling entity coreference systems might be to improve the efficiency of a single pairwise comparison. In other words, an entity coreference algorithm should compute the similarity between a pair of instances by only considering a selected portion of their context information and the key point here is how to perform such context selection appropriately. Accuracy and efficiency need to be balanced out well.

1.2 Contributions

Based upon the above discussions, this dissertation will summarize the accomplished research for building a domain-independent and scalable entity coreference system on the Semantic Web. The algorithms take heterogeneous Semantic Web data as input, collect instance context and appropriately utilize it to detect equivalent instances. This dissertation provides the following specific contributions to the research area of entity coreference in the Semantic Web:

1. Developed a mechanism for automatically collecting and weighting context information of ontology instances in a domain-independent manner. Based upon the collected and weighted context information, an entity coreference algorithm, *EPWNG*, was designed for detecting coreferent instances in the Semantic Web. *EPWNG* focuses on achieving high precision and recall by appropriately utilizing instance context information domain-independently and has been shown to outperform several state-of-the-art algorithms on small-scale benchmark datasets, with 1% to 4% higher F1-scores. Furthermore, a sampling and utility function based pruning algorithm, *U-EPWNG*, was proposed in order to speed up the computation for a single pair of instances. Our

experiments show that *U-EPWNG* provides a runtime speedup factor of 30 to 75 compared to *EPWNG*;

2. In order to further improve the scalability of the entire coreference process on large-scale Semantic Web datasets, an on-the-fly candidate selection algorithm, *P-EPWNG*, was proposed. *P-EPWNG* relies on the hypothesis that coreferent instances are also similar to the same set of other instances. The algorithm takes advantage of the matching histories of the instances in the datasets to other instances in order to prune instance pairs that are not likely to be coreferent. Compared to *EPWNG*, the overall entity coreference process was sped up by 2 to 3 orders of magnitude while only making a small sacrifice in the coreference F1-scores (at most 0.71% lower). Moreover, when compared against state-of-the-art candidate selection algorithms, our on-the-fly pruning technique leads to a speedup factor of 18 to 24 while maintaining competitive F1-scores (at most 0.66% lower).

3. To improve the coverage on true matches of the on-the-fly pruning algorithm, a pre-selection or offline candidate selection algorithm was further developed. This offline candidate selection technique is unsupervised and pre-selects candidate pairs by only comparing their partial context information. Such context is selected in an automatic and domain-independent manner without human intervention. Compared to *P-EPWNG*, this pre-selection technique achieves 1.7% to 4.5% higher coverage on true matches and enables the entire coreference process to run 24% to 64% faster. When compared against state-of-the-art candidate selection systems, the overall entity coreference process was sped up by a factor of 16 to 61 and, somewhat surprisingly, results in 0.3% to 0.5% higher F1-scores.

4. Instead of manually specifying predicate comparability, a value-based property matching technique was designed. This technique extracts tokens from the values of different properties and treats two properties to be comparable or matched if their token sets are sufficiently similar as measured by a modified version of the Jaccard similarity measure. By applying this property matching algorithm to the Billion Triples Challenge 2012 (BTC) dataset, which has tens of thousands of predicates, and by adopting our proposed *EPWNG* and *Offline* algorithms, our system outperforms other comparison and state-of-the-art systems, with about 7% higher F1-scores and a speedup factor of 66 on up to 2 million randomly selected instances from BTC.

5. The *EPWNG* algorithm has been applied to a project for cervical cancer patient classification, assisting cervical cancer screening. Instead of trying to find equivalent patients, in this project, we compute the similarities between a testing patient case and all training patient cases, and then utilize these similarity scores to determine the label of the testing case with majority vote. Utilizing our *Bag-Of-Chain* approach for similarity computation, by combining different types of information, such as basic patient information, patient clinical test results and patient digital images, our proposed system achieves 1.25% higher classification accuracy than the best accuracy of the other comparison systems. Computationally, the current system takes about 3 to 4 seconds for classifying a given testing case. The results demonstrate the potential of putting this technique into practical use.

1.3 Dissertation Overview

The rest of the dissertation will describe the details of all developed algorithms and is organized as following:

1. In Chapter 2, we will discuss the necessary background knowledge, including basic information about the Semantic Web, related work on Entity Coreference, research work about handling scalability issues for detecting coreference relationships for large-scale datasets. We will also talk about those frequently used evaluation metrics and well-adopted datasets for evaluating entity coreference systems.

2. In Chapter 3, we formally present our *EPWNG* algorithm for detecting coreference relationships between ontology instances. All details about the design choices of the algorithm will be discussed. Furthermore, we describe the sampling and utility function based pruning technique in Chapter 4 and compare it to *EPWNG*. The work here has been published in CIKM 2010 [16], JDIQ [18], and FLAIRS 2012 [19].

3. In the next two chapters, Chapter 5 and Chapter 6, we propose the two different candidate selection algorithms respectively. The details of the on-the-fly and offline algorithms will be presented and also they will be compared against state-of-the-art systems. These two algorithms correspond to my work published in Web Intelligence 2012 [20] and ISWC 2011 [21].

4. In Chapter 7, we describe how we modify the previous *EPWNG* and *Offline* algorithms for performing entity coreference on up to 2 million instances randomly selected from the Billion Triples Challenge 2012 dataset. And Chapter 8 presents how we apply our coreference algorithms for assisting cervical cancer screening. Although no actual

papers have been published, we have submitted a journal article describing this work to the IEEE Transactions on Medical Imaging.

5. Finally, we summarize the accomplished research and demonstrate the overall architecture of our proposed entity coreference system in Chapter 9. We also envision potential future work to further advance the state-of-the-art of building an interlinked data hub in the Semantic Web.

Chapter 2

Background and Related Work

In this chapter, I review important technology and discuss work related to the accomplished research. First, I provide a brief introduction to the Semantic Web and Linked Data. Furthermore, I discuss entity coreference in the settings of both free text and the Semantic Web, and describe existing techniques for building scalable entity coreference systems. Finally, I will introduce metrics and datasets that are frequently adopted for evaluating entity coreference algorithms.

2.1 Semantic Web

2.1.1 Knowledge Representation in the Semantic Web

Representing Data in the Semantic Web

RDF is a graph based data model for describing resources and their relationships in the Web. Although RDF is commonly described as a directed and labeled graph, as shown in Figure 2.1, many researchers prefer to think of it as a set of triples, each consisting of a subject, predicate and object in the form of <*subject, predicate, object*>. The subject is

the source of an edge, the predicate is its label, and the object is its target. The subject and predicate are always URIs, but the object can either be a URI or a literal. URIs that appear as objects in some triples can also be the subjects in other triples. The terms *subject* and *object* do not limit how the triples are viewed; by switching to another predicate which is the inverse of the current one, the subject and object of a triple can be exchanged. A literal could be plain, such as string, date, number, and even an arbitrary sequence of characters; or it could be adorned with XML Schema datatype information[1]. RDF defines a distinguished property called *rdf:type* to relate a subject to a class. For example, the following triple, *<http://rpi.edu/jah, rdf:type, http://www.example.com/university#professor>*, defines that "Jim Hendler", who is identified by the subject URI, is an instance of the professor class defined in some ontology.

In Figure 2.1, we show what an RDF graph actually looks like. In this example, we have several instances represented with ovals: *Person1, Paper1, Person2, and MIT*, and a couple of literal values drawn with rectangles, including "Tim Berners-Lee" and "Semantic Web". The edges are predicates that connect instances and literals. The meanings conveyed in the graph are intuitive but note that *Person2* links back to *Person1* via the *knows* predicate, making this a graph not just a tree.

Although essentially Semantic Web data are graphs as shown in Figure 2.1, various formats are available for representing the data in text. RDF/XML is one that represents data in the traditional XML format, which is also very verbose. N-Triples[2] and Turtle [3] take human readability into consideration for their design; and both formats are more compact compared to RDF/XML. By using N-Triples or Turtle, users simply need to write down

[1]http://www.w3.org/TR/xmlschema-formal/
[2]http://www.w3.org/2001/sw/RDFCore/ntriples/
[3]http://www.w3.org/TR/turtle/

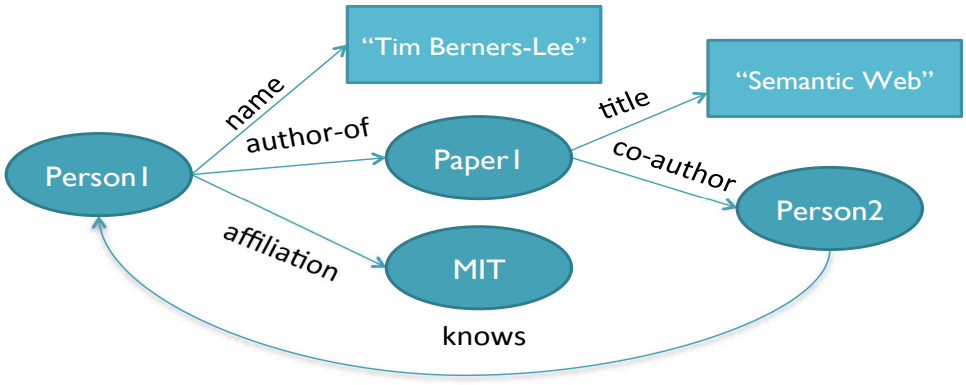


Figure 2.1: An example of an RDF graph

RDF graphs by using triples; and then tools, such as Jena and OWL API, can be used to load data in such formats to enable complicated queries. N-Quads[4] is another format and is very similar to N-Triples and Turtle. In addition to the traditional *subject, predicate and object* fields, an optional *context* field is added to indicate the provenance (the source) of each particular triple.

By adopting the Turtle format, in Figure 2.2, we show how to represent the above RDF graph. There are several basic rules in Turtle. First of all, URIs are embraced with <

```
Turtle:
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix skos: <http://www.w3.org/2004/02/skos/> .

<http://acm.org/~tbl> foaf:name "Tim Berners-Lee";
                      skos:author <http://scienicam.com/semantic-web>;
                      skos:affliation <http://www.mit.edu>.

<http://scienicam.com/semantic-web> skos:title "Semantic Web".
                                    skos:coauthor <http://rpi.edu/~jah>.

<http://rpi.edu/~jah> foaf:knows <http://acm.org/~tbl>.
```

Figure 2.2: An example of Representing Semantic Web Data in Turtle Format

and >, while literal values are always in quotes. Also, in Turtle, statements typically end

[4]http://sw.deri.org/2008/07/n-quads/

with the punctuation "." except for the following situation. Turtle allows users to ignore the subsequent subjects if triples that describe the same subject are grouped together; in this case, except for the last statement, the punctuation ";" is used instead of ".". As demonstrated in the above example, the first three statements are all about the subject *http://acm.org/-tbl* and therefore the second and the third subjects are ignored; also, instead of ending the two intermediate statements with ".", ";" is adopted. Moreover, in Turtle, we can use abbreviations of domain names throughout the whole document by defining prefixes at the beginning. In the given example, three prefixes are defined: *rdf, foaf and skos*, which are used for the rest of this little document.

Defining Ontologies

The goal of the Semantic Web is to automate machine processing of web documents by making their meanings explicit. To this end, Semantic Web researchers have developed languages and software that add explicit semantics to the content-structuring aspects of the Extensible Markup Language (XML)[5]. A Semantic Web language allows users to create ontologies [22], which specify standard terms and machine-readable definitions. Information resources (such as web pages and databases) then commit to one or more ontologies, thus stating which sets of definitions are applicable. Minimally, an ontology is an explicit and formal specification of a conceptualization, formally describing a domain of discourse. An ontology consists of a set of terms (classes) and their relationships (class hierarchies and predicates). For example, an academic ontology (that models universities, colleges, professors, students, courses, etc.) might state that both classes *Master student* and *Ph.D. student* are subclasses of the *Student* class and that they are disjoint, i.e., no single student

[5]http://www.w3.org/TR/xml/

can be a master student and PhD student at the same time. These definitions describe some of the meaning of the terms at schema level. Because ontologies adopt formal logic, the intended meaning of assertions using the vocabulary is unambiguous, and therefore, it avoids the ambiguities of natural language.

There are several different languages available in the Semantic Web. As intro ducted previously, RDF is a graph based data model for describing resources and their relationships in the Web; however, it does not enable users to represent any actual semantics. One step further, RDF Schema, or RDFS for short, can be thought of as a weakly expressive ontology language and is used to provide some basic semantics for the classes and properties. In particular, there are the properties *rdfs:subclassOf* and *rdfs:subpropertyOf*, which allow class and property hierarchies to be defined, and *rdfs:domain* and *rdfs:range* which define the classes of the subjects and objects of triples that use specific properties.

OWL is an ontology language designed specifically for the Web that is compatible with XML, as well as other W3C standards. On one hand, syntactically, an OWL ontology is a valid RDF document and a valid XML document. This enables ontologies and documents written in OWL to be processed by the wide range of XML and RDF tools that are already available, such as Jena [23] and OWL API [24]. On the other hand, OWL adds significant expressivity for describing the semantics of RDF vocabularies. OWL is based on description logics, a decidable fragment of first-order logic. Although most interesting description logics, including OWL, have worst-case exponential complexity, there exist algorithms, such as tableau-based reasoning, that are quite fast for typical ontologies. Description logics, and therefore OWL, have a set theoretic semantics. Classes are defined as subsets of the domain of interpretation (Δ^I) and properties are subsets of the Cartesian product $\Delta^I \times \Delta^I$. Mostly, the logic perspective of OWL is not relevant to the research in this dissertation but full

details can be found in several tutorials [25, 26, 27].

As a concrete example, Figure 2.3 demonstrates a little ontology that defines two classes and one property. At the beginning of this ontology, we have a "Person" class, followed by the definition of a property, called "hasChild". The surface forms of these two also convey their actual meanings. Note that the "hasChild" property is defined to connect person to person, specified by its domain and range. Making an analogy to what we have in Mathematics, these can be treated as the domain and range of a function. At the bottom of the ontology, we also define the "Parent" class, whose definition is a little more complex. To make things simple, the few lines of the definition simply say that a parent is whoever that has at least one child. Overall, the design of RDF and OWL by groups of international experts provided a set of standards for knowledge representation in the Semantic Web.

For the entity coreference problem to be addressed in this dissertation, the most relevant part to OWL is the ***owl:sameAs*** predicate, which is used to express equivalence between two ontology instances. This *owl:sameAs* predicate is transitive and symmetric, as demonstrated in Figures 2.4(a) and 2.4(b). Given instances a, b and c, if a is equivalent to b and b is equivalent to c, then a is equivalent to c, implied according to Transitivity. We will also be able to imply that b is equivalent to a, c is equivalent to b, and c is equivalent to a, all according to Symmetry. Practically, by using the *owl:sameAs* predicate, a user might specify that a person instance in the ACM database with the name *James Smith* is the same as another person named *J. Smith* in DBLP.

2.1.2 Linked Data

The Semantic Web is based on the idea that there are a number of ontologies, and different information resources commit to the definitions in these ontologies (see Figure 2.5). When

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#">

   <owl:Class rdf:ID="Person" />

  <owl:ObjectProperty rdf:ID="hasChild">
     <rdfs:domain rdf:resource="#Person/>
     <rdfs:range rdf:resource="#Person/>
   </owl:ObjectProperty>

   <owl:Class rdf:ID="Parent">
     <owl:equivalentClass>
       <owl:Restriction>
         <owl:onProperty rdf:resource="#hasChild">
         <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
           1
         </owl:minCardinality>
       </owl:Restriction>
     </owl:equivalentClass>
   </owl:Class>

</rdf:RDF>
```

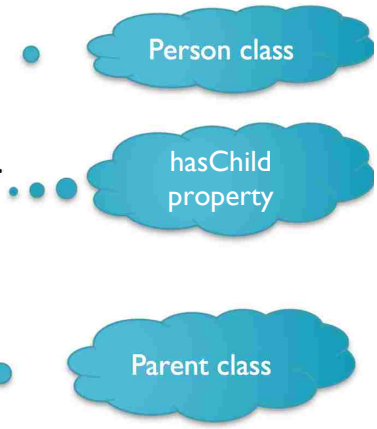


Figure 2.3: An Example of a Little Ontology

two sources commit to the same ontology (via *owl:imports*, or perhaps implicitly by simply using the ontology's namespace), then the same meaning is intended for any term from that ontology. This vision is decentralized in that any source can commit to any ontology, and any source can create a new ontology. Ontologies can be very source-specific, being constructed for a particular application without consideration of future integration.

In the Semantic Web, Linked Data is about using the Web to connect related data that wasn't previously linked, or using the Web to lower the barriers to linking data currently linked using other methods. For example, in Figure 2.5, the dashed lines between data sources $D4$ and $D6$ represent the coreference relationships between their instances. Quoting from Christian Bizer, Tom Heath and Tim Berners-Lee [4]:

```
Turtle:
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<http://www.example.org/a> owl:sameAs <http://www.example.org/b> .
<http://www.example.org/b> owl:sameAs <http://www.example.org/c> .
```

(a) Explicit owl:sameAs Statements

```
Turtle:
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<http://www.example.org/b> owl:sameAs <http://www.example.org/a> .
<http://www.example.org/c> owl:sameAs <http://www.example.org/b> .

<http://www.example.org/a> owl:sameAs <http://www.example.org/c> .
<http://www.example.org/c> owl:sameAs <http://www.example.org/a> .
```

(b) Implied owl:sameAs Statements

Figure 2.4: An Example of owl:sameAs Statements

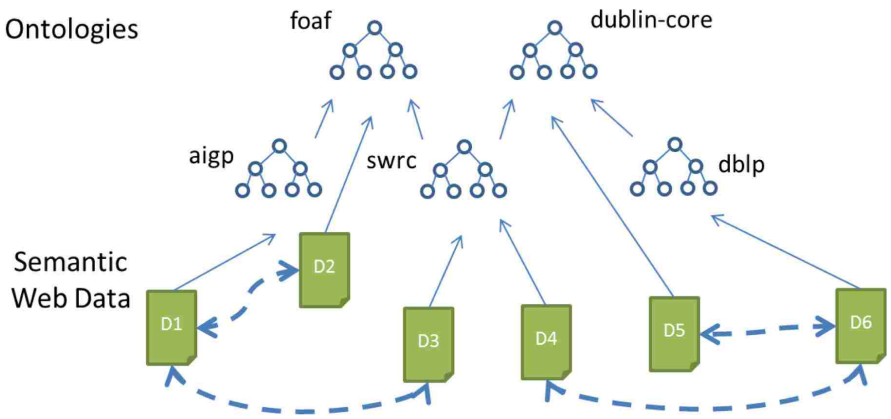


Figure 2.5: An example diagram of the relationships between ontologies and Semantic Web Data Sources

linked data describes a method of publishing structured data so that it can be interlinked and become more useful. It builds upon standard Web technologies such as HTTP and URIs, but rather than using them to serve web pages for human

readers, it extends them to share information in a way that can be read automatically by computers. This enables data from different sources to be connected and queried.

As quoted above, Linked Data is not only about exposing data using Web technologies, nor is it simply an elegant way to solve interoperability issues. Linked data is fundamentally about helping the web to transition from a web of documents to a data web. Tim Berners-Lee set several principles about Linked Data[6]. Among these, the most important ones to me are: 1) Use HTTP URIs to name things so that people can look up those things; 2) Include Links to other URIs, so that users can discover more things. Adopting the first principle enables entities or things to be identified and allows users to be able to find them on the Web. The second principle is really significant in the sense that it meets the nature of the Web and the Semantic Web where information is stored in a distributed manner in many different places, and linkages from one identifier in one data source to identifiers in other data sources give users the opportunity to browse and obtain more information. Take the following as an example. If a single author has published in two different conferences, and both of them expose their paper information as Linked Data, and DBpedia has the biographical information of the author, an application can then easily mesh these information all together with a simple query automatically, as if all the data was in a single database.

Currently, there already exist hundreds of different datasets on the Web, published according to the Linked Data principles: hundreds of millions of different identifiers (ontology instances) exist and people can rely on these instances to grab data about various kinds of things, such as people, locations, organizations, books, movies, musics, actors, cities, and a lot more. In other words, these datasets (from different domains and of various scales) form a huge and global web-scale database that could be adopted to fuel real-world applications

[6]http://www.w3.org/DesignIssues/LinkedData.html

in many different fields.

In recent years, the Open Data Movement that aims to release huge data sets, embraced the Linked Data technologies and best practices to publish a plethora of different interlinked data sets on the web. These datasets are usually from government authorities, such as data.gov, data.gov.uk, etc. In general, these datasets are published with an open license and are made available for everyone to use and to republish without restrictions from various issues, including copyright, patents or others. When combined with Linked Data, we finally have Linked Open Data, where data are publicly available and well interlinked to each other.

The most notable effort in this area is the Linking Open Data project[7]. After nearly 6 years of fast development, it currently hosts 295 datasets. The current statistics are collected by Richard Cyganiak and Anja Jentzsch and are demonstrated with the well-known Linked Open Data cloud diagram as shown in Figure 2.6. They update the diagram periodically, trying to capture the latest status of all published data sets with their sizes and inter-links. The diagram is automatically sketched using the CKAN API to get JSON for each of the data sets and then manually clustered and colored by their characteristics.

One can also add his/her own datasets so that they will be included in the next version of the diagram. Several conditions need to be satisfied before the data can be included. Among these, the most important criteria are: 1) The data must be resolvable via *http://* URIs; 2) The dataset should contain at least 1,000 triples; 3) At least 50 links should exist between the dataset to be added and other existing datasets already in the cloud. The full set of criteria can be viewed from their website[8]. Qualified datasets can be added to Data Hub[9], the open registry of data and content packages, and will be gone through the rest of

[7]http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[8]http://lod-cloud.net/
[9]http://datahub.io/

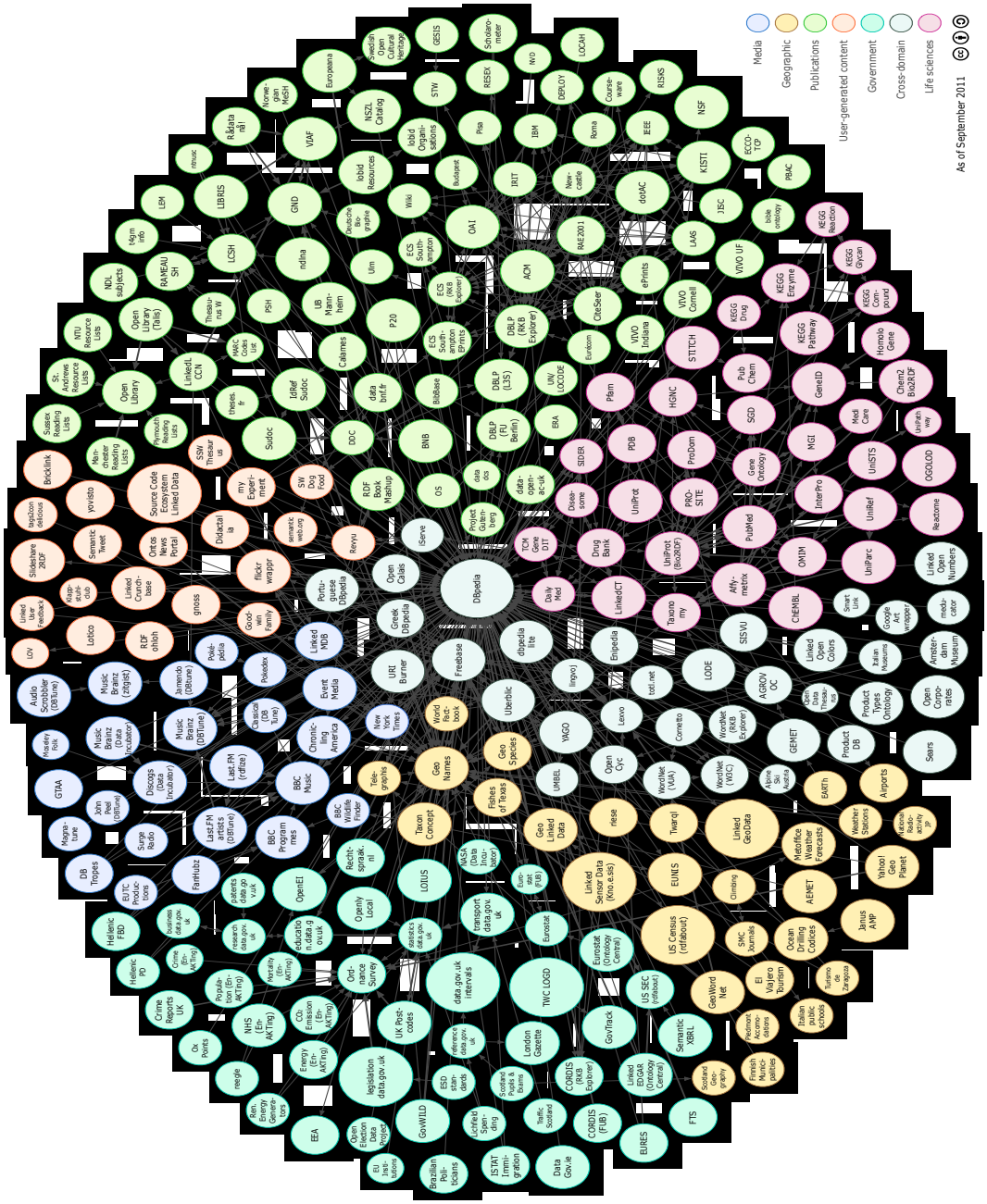


Figure 2.6: The Linked Open Data Cloud Diagram (as of 09/19/2011)

the process.

Commercial giants, such as Google, Facebook and Microsoft, have already adopted some of the principles behind the Semantic Web and Linked Data, such as Google's Knowledge

Graph[10], Facebook's Open Graph Protocol[11] and Schema.org[12]. The idea behind Google's Knowledge Graph is that there are nodes that represent distinct entities and also relationships between these entities; and such nodes and their relationships actually form a graph. The nodes and relationships can be borrowed from existing structured datasets (e.g., Freebase) but are also continuously extracted from the Web, merged or added to the existing graph. To me, RDF can be adopted for representing these information in the Knowledge Graph and other research outcome of the Semantic Web field could also be applied, e.g., scalable reasoning. Schema.org holds a collection of schemas or html tags that data publishers could use to label things or entities they describe on the web. Because these tags are recognized by major search engines, the meanings (presumably pre-agreed between search engines) can be interpreted in order to provide better search results. This is like a set of commonly used ontologies where classes and predicates are defined and aligned across these ontologies to provide common semantics. All these efforts from industry show that the Semantic Web and linked data is a new way of grabbing and utilizing structured data in their own platforms. Moreover, semantic markup standards, such as Microformats[13], Microdata[14], and RDFa[15], all allow structured data to be embedded into traditional HTML pages. This pushes millions of pages containing structured data on to the Web, and all these data can be linked and consumed for specific applications.

Although we do see the current fast growth and the potentially bright future of Linked Data, we need to realize that a lot of these potentials rely on the existence of interlinked datasets. Without the linkages across different datasets, systems will not be able to mesh

[10]http://www.google.com/insidesearch/features/search/knowledge.html
[11]http://ogp.me/
[12]http://schema.org/
[13]http://microformats.org/
[14]http://www.whatwg.org/specs/web-apps/current-work/#microdata
[15]http://www.w3.org/TR/xhtml-rdfa-primer/

all information about the author from various sources together, and therefore end users will still have to query those data sources individually to obtain a comprehensive description of that author. Overall, one important issue in publishing datasets as Linked Data and really utilizing Linked Data is to discover the links amongst the datasets. Such links can either be manually identified or generated automatically. Manually identifying the links could work for small datasets; however, automatic and scalable approaches are needed to manage large-scale datasets. Such needs for interlinking heterogeneous and large-scale datasets in the Semantic Web open the door for a new research direction. Before we dive into the details of the accomplished dissertation research, in the rest of this chapter, we will firstly review the current literature of entity coreference from different research fields and also discuss the pros and cons of contemporary approaches.

2.2 A Brief Overview of Information Retrieval and Machine Learning Techniques

Here, we briefly discuss information retrieval and machine learning techniques, since machine learning is closely related to some previous work and our research adopts information retrieval techniques for improving the scalability of our entity coreference system.

2.2.1 Information Retrieval

Given a large collection of documents, Information Retrieval is a traditional technique that allows users to quickly look up documents that contain some given words. Instead of having to traverse through the whole document collection, an inverted index is built by pre-processing all documents in the collection to speed up the query process. Figure 2.7 demonstrates an example of an inverted index.

Document ID	Content
1	Dezhao Song studied at Lehigh University
2	He graduated in the year of 2013 from Lehigh University
3	Dezhao performed research at Lehigh

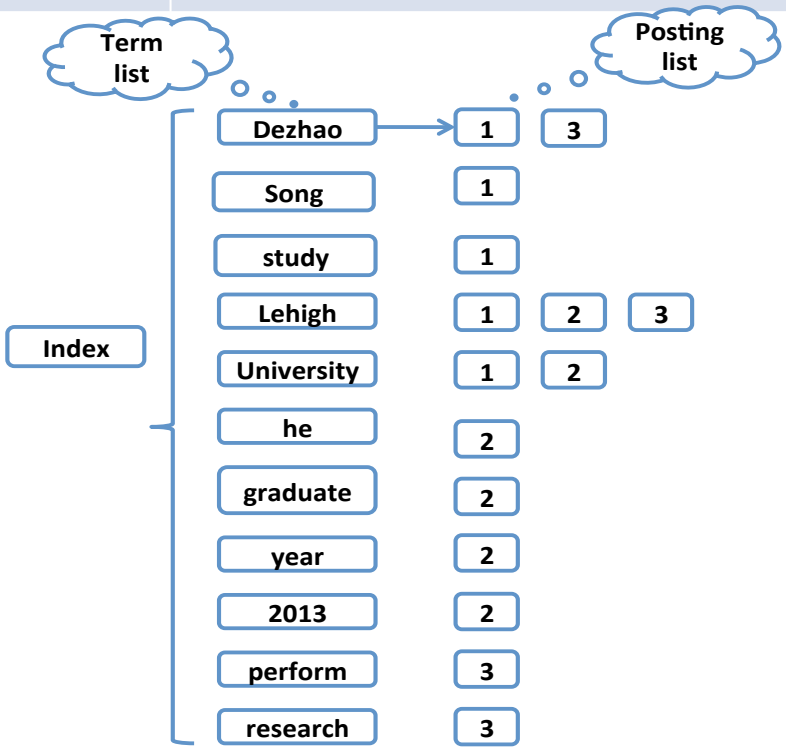


Figure 2.7: An Example of Inverted Index

From the upper part of this example, first of all, we see three documents, a document collection, identified with ID 1 to 3 respectively; and each document has their own contents. Furthermore, the lower part of this diagram shows the built inverted index, with two primary components: Term List and Posting List. The Term List contains all tokens extracted from the contents of all documents with stop words filtered out. Each term in the term list is then associated with a posting list that includes all documents that have this term or token in their contents. Specifically, in our example, "Dezhao" is a term and it appears in both Document 1 and 3; therefore, the posting list of the "Dezhao" term has both documents in it. Please note that some tokens, such as "in", "the", and "of", were not included in

the term list, since they frequently appear in many documents (in real world scenarios) and are not sufficiently representing a document. Also, sometimes, a token in the content is changed to another form. For example, in Figure 2.7, the token "graduated" was changed to its original form "graduate" when building the index. This is typically referred to as Stemming, a pre-processing or normalization step when building inverted index.

With this inverted index, users could easily perform fast index look-up to find documents that satisfy certain constraints. As a concrete example, suppose we have the following query: *Find documents that contain term "Dezhao" and term "Lehigh"*. Given this query, we first obtain the posting lists of both terms: Document 1 and 3 for term "Dezhao" and Document 1, 2, and 3 for term "Lehigh". Next, we perform an intersection of the two posting lists and finally have the answer to the query to be Document 1 and 3. Disjunctive queries can be performed in a similar way; instead of doing intersection, a union will be performed to merge the posting lists of different terms from the query. According to the literature of information retrieval research, techniques have been developed for index optimization, such as index compression and adding location information into the index to support more types of queries [28].

2.2.2 Machine Learning

Machine Learning is a well known technique for performing prediction by utilizing from known information and knowledge. According to Alpaydin [29], "Machine learning is programming computers to optimize a performance criterion using example data of past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive

to gain knowledge from data, or both." Overall, machine learning is to use existing data and knowledge to predict some currently unknown. There are primarily three types of machine learning techniques: Supervised Learning, Unsupervised Learning and Reinforcement Learning.

For supervised learning, in general, we have training data and a mathematical model that actually uses the training data to learn the parameters in the model. For a given machine learning problem, we also need to define a set of features that are specifically designed for a given problem. From the training data, values of the defined features are extracted and are utilized by the mathematical model to learn the values of the parameters. To evaluate machine learning algorithms, we also have testing data where we extract values for the same set of features and let the learned mathematical model to do prediction by utilizing the learned parameter values. Models for such approaches include Naive Bayes, Decision Tree, Random Forest, etc. As a concrete example, we could try to predict if a customer would buy diaper by considering several factors or features: 1) Does this customer have a baby; 2) Gender of this customer; and 3) Is there a football game tonight? With these features, we probably be able to learn the probability that a customer buys diaper under each feature and combine them together for prediction. Note all these probabilities are learned from the training data automatically.

Different from supervised approaches, there are also unsupervised approaches. Instead of trying exploit training data or labelled data, such approaches operate directly on the testing data and try to discover the structure of the data. Clustering algorithms would be good examples of such category. The advantage of unsupervised approaches is that they do not require any labeled data which could be difficult to obtain for certain domains. However, because there is not any formal training process, unsupervised approaches can only rely on

some similarity and distance measures for prediction.

Finally, reinforcement learning, different from previous two types of machine learning algorithms, is concerned with how intelligent agents should act in an environment to maximize some notion of reward. The agent executes actions which cause the observable state of the environment to change. Through a sequence of actions, the agent attempts to gather knowledge about how the environment responds to its actions, and attempts to synthesize a sequence of actions that maximizes a cumulative reward. Different from supervised learning, reinforcement learning based algorithms learn how to perform actions by considering the outcome of previous actions in order to achieve the highest reward.

2.3 Entity Coreference

Entity coreference has drawn interests from researchers in a variety of fields, including Natural Language Processing, Database and the Semantic Web. The purpose of entity coreference[16] is to determine if syntactically distinct or identical identifiers refer to the same real world entity. An identifier is a string, such as a person name, a publication title, a geographical location name, a company name, a URI, etc. Identifiers can appear in free text, such as news articles and web pages, or in structured and semi-structured data sources, including databases and the Semantic Web.

In free text, entity coreference is to decide which name mentions actually represent the same real world entity. In the following example, coreference algorithms may be used to determine if the pronoun *he* is referring to the name *Dezhao Song*.

> *Dezhao Song* is a fifth-year Ph.D. student and *he* is working in the SWAT

[16]Entity coreference is also referred to as deduplication [7], entity disambiguation [15], etc.

lab of *Lehigh University*. For more details, please consult his LU website.

Moreover, entity coreference also detects equivalences between named entities. Here named entities mean those identifiers that have been explicitly assigned a name, such as a person name, a publication title, etc. In the above example, the name *Lehigh University* is actually coreferent with LU, since the latter is an abbreviation of the former given this particular context. In databases, entity coreference is better known as record linkage or deduplication and is used to detect duplicate database records where such records may come from heterogeneous data sources and may even be represented with different schemas.

With the development of Semantic Web technology, there is rapidly growing interest on entity coreference in the community. In the Semantic Web, entity coreference can happen between a free text mention and an ontology instance or between ontology instances themselves. The latter has received more attention from the research community, since being able to automatically provide high quality *owl:sameAs* links between heterogeneous and large-scale datasets is recognized as one critical step toward transitioning the current Web from a web of documents to an interlinked data web.

Before we start discussing the relevant techniques, let's define some terminologies that will be used throughout the dissertation:

- Identifier/mention [30, 31, 32, 33]. An identifier or a mention is a string that appears in textual content and is a reference of a real world entity. For example, a person name that appears in a free text document represents a real world person; the primary key of database record represents this record; a URI of an ontology instance identifies this instance in a particular RDF graph. Broadly, an identifier represents all aspects of a reference in a given context.

- Entity [30, 31, 32, 33]. In the rest of this dissertation, we use the term *entity* to represent a real world entity. For example, *Dezhao Song*, the fifth-year Ph.D. student at Lehigh University, Bethlehem, Pennsylvania, U.S.A. is an entity. The paper titled *Domain Independent Entity Coreference in RDF Graphs* and published in the CIKM2010 conference is another entity.

- Coreferent [31, 32] or Coreferential [30, 33]. Identifiers that refer to the same entity are said to be coreferent. For instance, the following two syntactically distinct URIs are from Semantic Web Dog Food and DBLP respectively while they both represent the same real world person.

  – http://data.semanticweb.org/person/dezhao-song/html

  – http://www.informatik.uni-trier.de/ ley/db/indices/a-tree/s/Song:Dezhao.html

2.3.1 Standard Evaluation Techniques for Entity Coreference

In this section, I will describe the well-adopted metrics and datasets for evaluating entity coreference systems. I will also give some comments on these current evaluation metrics and propose a few more options.

Evaluation Metrics for Entity Coreference Systems

In the literature of entity coreference research, three metrics have been well adopted for evaluating entity coreference systems [13, 34, 35, 36]: Precision, Recall and F1-score as computed in Equations 2.1 to 2.3. Precision is measured as the number of correctly detected coreferent pairs divided by the total number of detected pairs given a threshold t; Recall is defined as the number of correctly detected coreferent pairs divided by the total number of coreferent pairs given a set of ontology instances; F1-score gives a comprehensive view of

how well a system performs:

$$Precision_t = \frac{|correctly\ detected|}{|all\ detected|} \tag{2.1}$$

$$Recall_t = \frac{|correctly\ detected|}{|true\ matches|} \tag{2.2}$$

$$F1 - Score_t = 2 * \frac{Precision_t * Recall_t}{Precision_t + Recall_t} \tag{2.3}$$

where t represents threshold in all the above three equations.

Since it could be difficult to obtain perfect groundtruth for large datasets, sampled precision (sP) and relative recall ($relR$) could be adopted. $relR$ is calculated with Equation 2.4:

$$relR = \frac{|correctly\ detected\ pairs\ from\ one\ system|}{|correctly\ detected\ pairs\ from\ all\ systems|} \tag{2.4}$$

To measure sP, we can manually check the correctness of a subset of the detected links. The idea of *wisdom of the crowd* that we discussed previously can be adopted for assessing precision, such as Amazon Mechanical Turk; however, having perfect groundtruth to measure recall could still be challenging.

In some previous works, the metric accuracy was also adopted [37, 38, 39], which is formally defined in Equation 2.5:

$$Accuracy = \frac{|correctly\ detected\ coreferent\ pairs| + |correctly\ detected\ non\text{–}coreferent\ pairs|}{|all\ tested\ instance\ pairs|} \tag{2.5}$$

where we have a number of instance pairs that need to be determined as coreferent or non-coreferent; and the nominator here is the number of instance pairs that are correctly

labeled by an algorithm.

Evaluating Blocking/Candidate Selection Techniques

As we will discuss in more details in Section 2.4, candidate selection technique is to scale an entity coreference algorithm to large scale datasets. Instead of comparing every pair of instances between two datasets. some pairs of instances will be selected and more expensive coreference algorithms will only be applied to these selected pairs that are more likely to be coreferent than others in order to reduce the overall computational cost.

To evaluate candidate selection algorithms, three traditional metrics have been frequently used [40, 41, 42, 43, 44]: Pairwise Completeness (PC), Reduction Ratio (RR) and $F1\text{-}score_{cs}$ (F_{cs}) as shown in Equations 2.6 to 2.8. PC and RR evaluate how many true positives are retained by a candidate selection algorithm and the degree to which it reduces the number of pairwise comparisons needed respectively; and F_{cs} is the F1-score of PC and RR, giving a comprehensive view of how well a candidate selection algorithm performs:

$$PC = \frac{|true\ matches\ in\ candidate\ set|}{|true\ matches|} \tag{2.6}$$

$$RR = 1 - \frac{|candidate\ set|}{N * M} \tag{2.7}$$

$$F_{cs} = 2 * \frac{PC * RR}{PC + RR} \tag{2.8}$$

where N and M are the sizes of two instance sets that are matched to one another. The preselector should have a high PC so that most of the true matches (coreferent instance pairs) will be included in the candidate set and sent to the resolver. In the meanwhile, RR is also important, since a candidate selection algorithm also needs to be able to reduce as many instance pairs as possible to save the overall computational cost.

Another important metric is the runtime of both candidate selection and the entire entity coreference process. Due to the large scale of Semantic Web data today, we need entity coreference systems that could really scale to large datasets (e.g., datasets with millions of instances). Since candidate selection techniques are designed to scale entity coreference algorithms, such techniques themselves need to scale to large datasets as well. A balance between RR and PC is particularly important. Algorithms that only try to maximize one aspect are not acceptable. One thing to note is that according to Equation 2.7, a large change in the size of the candidate set may only be reflected by a small change in the RR due to its large denominator. Therefore, there is the need to adopt new evaluation methods and metrics to perform a more fair comparison between different systems. One option would be to apply the entity coreference resolver to the selected candidate pairs to: 1) measure the runtime of the entire process, including both candidate selection and entity coreference; 2) check how the missing true matches can affect the final coreference results. It is possible that even if those missing pairs were selected, the resolver would still not be able to detect them. Furthermore, in order to cover the last few missing true matches, more false positives could be selected, which would potentially add more computational complexity to the entire process.

Evaluation Datasets

Another important aspect of the evaluation is about datasets. In this section, we will introduce the frequently adopted datasets for evaluating entity coreference systems.

Freetext and Semi-structured Entity Coreference Datasets

John Smith. This dataset[17] was developed by Bagga and Baldwin with groundtruth

[17]http://alias-i.com/lingpipe/demos/data/johnSmith.tar.gz

provided and adopted for cross-document entity coreference [5, 14]. It consists of 197 articles from the 1996 and 1997 editions of New York Times, describing people named *John Smith*. All articles contain either the exact name *John Smith* or some variation. Each article mentions a single John Smith, i.e., all the names *John Smith* in the same article are coreferent. One disadvantage is that, in total, there are only 197 articles. In order to provide convincing experiment results, more data would be needed. Furthermore, with more data, it would allow researchers to test the scalability of their systems.

Web People Search Task. The Web People Search Task [45] provides a dataset for disambiguation of person names from web pages. Finding people and their information in the World Wide Web is one of the most common activities of Internet users; Person names, however, are highly ambiguous[18] [46]. In most cases, therefore, the results are a mixture of pages about different people that share the same name. The participating systems use web pages retrieved from a web search engine (queried with a given person name) as input, determine how many referents (different people) exist for a given person name, and assign to each referent its corresponding documents. This dataset includes 79 distinct person names, and 100 web pages are collected for each name with groundtruth.

DBWorld. This semi-structured dataset consists of 20 DBWorld posts[19] [15]. A DBWorld post typically contains an introduction of the upcoming event, topics of interest, important dates and a list of committee members and their affiliation information. The general layout of the DBWorld post is rarely consistent in terms of its structure. For example, sometimes the committee members of a conference are listed with their affiliation information; while in some other situations, they are listed with more information, such as country names. This variety gives the opportunity to entity coreference systems to explore

[18]http://www.census.gov/genealogy/www/data/1990surnames/index.html
[19]http://lsdis.cs.uga.edu/∼aleman/research/dbworlddis

how to utilize incomplete and/or heterogeneous information, which is common in general. Groundtruth is provided where each person named in the dataset links to a URI in an ontology populated from the DBLP bibliography. One issue of this dataset is that it only contains 20 DBWorld posts with 758 entities; and, generating the groundtruth for larger datasets can take a great amount of efforts.

CORA. Andrew McCallum created the CORA dataset[20]. The dataset contains research paper citations with labeled segments, including authors, title, affiliation, venue, date, page numbers and so on. Overall, 1,349 citations to 134 different research papers were included. Weis et al. added some more duplicates of the research papers with incomplete information [21], giving a little larger datasets containing 1878 citations.

Entity Coreference Datasets in the Semantic Web

As I will present later, in my current work, three datasets: RKB person and publication[22] [17] and SWAT person[23], were used for evaluating the entity coreference algorithm. These datasets contain up to three million instances, which is a reasonable scale to demonstrate the scalability of the proposed system. However, these datasets generally fall under the academia domain; in other words, they primarily describe researchers, academic publications, research institutions, etc. Therefore, it is important to find other datasets that have more diverse types of instances and are outside of the academia domain. Furthermore, the groundtruth information needs to be provided along with the datasets in order to evaluate our proposed algorithms, which means the datasets should have the coreference relationships between instances either explicitly or implicitly stated. Manually labeling

[20]http://www.cs.umass.edu/∼mccallum/data/cora-ie.tar.gz

[21]http://www.hpi.uni-potsdam.de/fileadmin/hpi/FG_Naumann/projekte/repeatability/CORA/cora-all-id.xml

[22]http://www.rkbexplorer.com/data/

[23]http://swat.cse.lehigh.edu/resources/data/

groundtruth for large-scale datasets is time-consuming and may not be feasible considering the current scale of the Semantic Web data today.

To my knowledge, currently, there are a few datasets that satisfy such requirements: DBpedia [47, 48], New York Times and Freebase [49]. DBpedia is typically treated as a Semantic Web version of the ordinary Wikipedia. It converts free text from Wikipedia pages to its Semantic Web representation. New York Times now provides an RDF version of some of its data about people, organizations and locations. Freebase has a more diverse variety of instance types in its data, ranging from science to products and to arts and entertainment. One important point is that these three datasets provide the needed groundtruth information across themselves. For example, New York Times and Freebase describe the U.S. actor Nicolas Cage with two distinct URIs and connect them with an *owl:sameAs* statement.

Furthermore, the Ontology Alignment Evaluation Initiative (OAEI)[24] includes an instance matching track that provides several benchmark datasets and the necessary groundtruth information. Some of the datasets are primarily targeting on comparing the precision and recall of the participating systems with only a few thousands instances, such as the Person1, Person2 and Restaurant datasets in the PR track of OAEI2010 [50]. However, some larger datasets are also provided.

Finally, we want to introduce the Billion Triples Challenge dataset which was also adopted in our evaluations. The BTC 2012 dataset was crawled from the web and consists of several subsets: Datahub[25], DBpedia, Freebase, Timbl (Tim Berners-Lee), and rest (i.e., others). In general, the URIs in each subset were used as seeds for crawling with different levels of expansion. For DBpedia and Freebase, no links were expanded; while rest, Datahub, and Timbl were expanded to level 2, 4, and 6 respectively to get more triples. We summarize

[24]http://oaei.ontologymatching.org/
[25]http://datahub.io/dataset

Table 2.1: Billion Triples Challenge Dataset Statistics

Number of Triples	1.4 Billion
Number of Instances	183 Million
Number of Predicates	57,000
Number of Classes	296,000

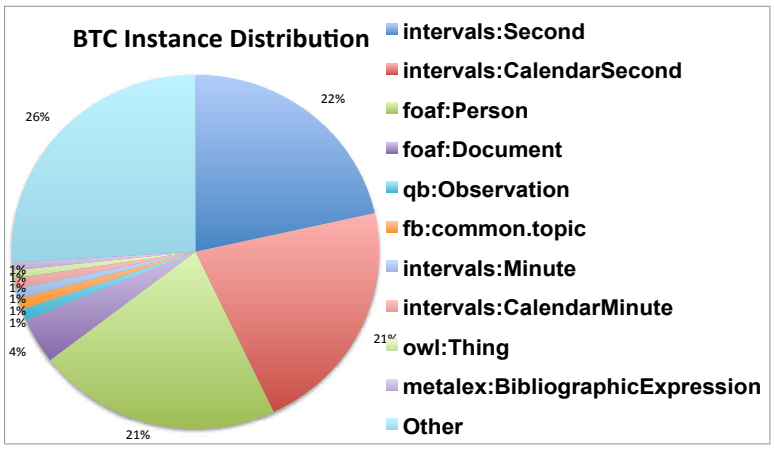


Figure 2.8: Distribution of Ontology Instances of Explicit Classes

the basic statistics of this dataset in Table 2.1. We also show the distribution of instances among the total 296K classes in Figure 2.8.

First of all, from Table 2.1, we can see that there are 57K predicates in this dataset, thus making the BTC dataset appropriate for testing predicate matching algorithms to see if the produced predicate mappings will enable an coreference algorithm to achieve decent coreference results. Manually linking predicates on this scale would be impossible. Furthermore, given the amount of the instances in this BTC dataset, it provides a perfect testbed to study whether entity coreference algorithms will be able to provide precise and comprehensive coreferent mappings in a scalable manner.

2.3.2 Word Sense Disambiguation and Database Deduplication

Word sense disambiguation (WSD) and duplicate record detection in databases are two closely related topics to entity coreference. A word can have multiple meanings while the

task of WSD is to choose the most appropriate one based upon the word's context [51, 52], such as a piece of free text. Duplicate record detection [7] or Merge/Purge [53] is to detect duplicate tuples and remove redundancies from databases. Different database records can give the same information but are distinct in their representations. For example, different records can represent a person's name differently, in the forms of full name or first initial plus family name.

Dong et. al. [54] proposed an entity coreference algorithm that exploits the relationships between different entities to improve system performance. They collectively resolve entities of multiple types via relational evidence propagation in dependency graphs. They applied the algorithm to multiple real world datasets and demonstrated its effectiveness. Kalashnikov and Mehrotra [55] proposed RELDC (Relational-based Data Cleaning), detecting coreferent mentions by analyzing entity relationships. The mentions and their relationships are viewed as a graph where edges represent the relationships between mentions. In general, instead of only using author names and affiliation information for disambiguation, each author identifier is enhanced with co-author information and others as well. As a concrete example from the paper, suppose we have three mentions named "D. White", "Dave White" and "Don White" respectively, and we want to decide which of three mentions are actually coreferent. From their surface names, it would be difficult to judge since they have the same initial and last name. However, with the assistance of the relational graph, it was decided that "Don White" is actually coreferent with "D. White", because: 1) Both of them have a co-author from MIT, while "Dave White" never had a MIT co-author; 2) Both mentions have similar titles for their publications. In order to scale to large graphs, certain optimization techniques were adopted, such as limiting the length of the relationships, constraining the number of relationships to be considered, controlling runtime, etc. They demonstrated

the effectiveness of their algorithm on real world datasets in different domains: author, publication and movie. As a combination of collective and relational based approaches, Bhattacharya and Getoor [56] developed a collective relational clustering algorithm that uses both attribute and relational information to detect coreferent entities. Their algorithm achieved 0.2% to 0.4% higher F1-score and ran about 100 times faster than state-of-the-art systems on two real world datasets. Different from the methods discussed above, Ioannou et. al. [57] proposed a novel framework for entity coreference with uncertainty, trying to detect coreferent instances at query-time. In their system, possible coreferent relationships are stored together with the data with some probability; then a novel query technique is employed for answering queries by considering such probabilities. Their system achieves better F1-score when comparing to some offline entity coreference systems on two datasets.

2.3.3 Entity Coreference in Free Text

Many researchers have been working on entity coreference in free text. Gooi and Allan [14] adopted the *Bag-of-Words* [58] approach to collect context information for resolving coreference relationships between person names in free text. In the *Bag-of-Words* model, a text (such as a sentence or a document) is represented as an unordered collection of words, disregarding grammar and word ordering. It is commonly used for document classification, where the frequency of each word is used as a feature for training a classifier. The authors employed different models, the incremental and agglomerative vector space models and KL divergence. When collecting context information, for each person name, they chose a window of 55 words centered on this name and used these words as context, called a *snippet*. The 55-word range may actually go across multiple sentences. They chose the number of 55 because it appeared to give the best results based upon their experiments, which is actually

one potential issue of this approach. For different datasets, the best window size may vary and it could be very time-consuming to determine the appropriate threshold by trying out those possible values in a brute-force manner. Pedersen et al. [59] and Hatzivassiloglou et al. [60] chose a similar way of locating context information. In both of these works, a window size of 50 words centered on the name mention was used.

Bagga and Baldwin [5] tried to resolve coreference relationships between person mentions with the name *John Smith* across different documents. The name of an identifier could either be *John Smith* or some variation with a middle initial/name, such as *John A. Smith*. In this work, the context information (a bag of words) is collected as follows. Their system first sends the documents to the CAMP system [61], a within document entity coreference system for free text. CAMP will build the coreference chains for each of the input documents. In the following example, the names *John Perry* and *He* will be output as coreferent. Note that CAMP did not only resolve coreferences on named mentions but also pronouns.

> *John Perry, of Weston Golf Club, announced his resignation yesterday. He was the President of the Massachusetts Golf Association.*

In the next step, another module, called SentenceExtractor, was adopted. For each name mention, it will extract all the sentences that contain this name or sentences that include other names that are coreferent to it. In other words, the SentenceExtractor produces a *summary* for each group of coreferent name mentions. The extracted summaries are then used for cross-document coreference. Instead of picking a pre-defined range of sentences or words as context information, in this work, the within-document coreference system, CAMP, helped to identify and restrict the range for collecting context information. However, it may depend on CAMP to perform well in order to collect high quality context. For example, if CAMP would have either low precision or low recall, the collected sentences and thus the

context information could be restrictive or noisy. Following a very similar fashion, Chen and Martin [62] used the EXERT[26] system for named entity detection and collecting context, therefore their system may also suffer from the same problem.

Mann and Yarowsky utilize an unsupervised clustering technique over a feature space for person name coreference [37]. Different from the previous approaches, they extracted more representative information from web pages, such as biographical information, marriage and parent/child relationships and so on. The general idea is that the system learns biographic facts by adopting some patterns that are learned with bootstrapping. With some seeds, such as (Mozart, 1756), a web query is firstly issued to retrieve matching web pages. Then, in these returned web pages, sentences and their substrings in which this seed appears in nearby association (e.g., Mozart was born in 1756) will be extracted. Finally, these extracted substrings will be generalized to produce patterns. For instance, *Mozart* will be replaced by <name>, and *1756* will be replaced by <birth year>. One example of the extracted patterns will look like this:

- <name> (<birth year> - ####)

Each # represents a digit, and thus <birth year> needs to consist of four digits. These patterns are then used to extract the needed information from freetext for coreference.

Han et al. deploy two supervised models, the Naive Bayes classifier and Support Vector Machine (SVM) [38], to disambiguate author names in citations [39]. Given a citation, their algorithm predicts if it is written by some author. Both classifiers have achieved reasonable performance in terms of accuracy. One problem is that their approaches worked fine with data from particular domains or types of data, such as academic publications and person mentions. Switching to other domains or entity types, new features need to be identified,

[26]http://sds.colorado.edu/EXERT

which sometimes requires specific domain expertise to design elegant features. Some graph based approaches have been employed as well to disambiguate mentions in social networks [63] and emails [64]. Other than pure free text, Wikipedia and Encyclopedia have also been used to find context information [65, 66]. Special types of Wikipedia pages (e.g., disambiguation pages) and the embedded hyperlinks have been employed for exploiting context information.

Named entity recognition [67] can be treated as a pre-processing step for entity coreference. It recognizes different types of mentions, such as person, organization, etc. Having such type information could be helpful for disambiguation. For example, when a person and a company share very similar names, their similarity scores may be very high; however, since their types are actually disjoint, i.e., no single instance can be a person and a company at the same time, by utilizing such information, an entity coreference algorithm should be able to differentiate them appropriately. The details of named entity recognition techniques are out of the scope of this dissertation.

2.3.4 Interlinking Semantic Web Data

String Matching based Instance Matching

With the emergence of the Semantic Web technologies, researchers have started showing interests in the entity coreference problem on the Semantic Web, essentially trying to realize the Linked Data vision. Hassel, et al. [15] proposed an ontology-driven disambiguation algorithm to match ontology instances created from the DBLP bibliography [3] to recognized named entity mentions in DBWorld documents[27]. They use the information provided in

[27]http://www.cs.wisc.edu/dbworld/

the triples of an ontology instance to match the context in free text. In this paper, the authors adopt different types of features for coreference, such as *Text-Proximity Relationships* and *Text Co-occurrence*. For *Text-Proximity Relationships*, in the ontology, one such relationship is affiliation. For example, if a person ontology instance, named *John Smith*, has affiliation information of *Stanford University* and in a DBWorld document, *John Smith* and *Stanford University* occur close to each other, then this adds some confidence that this person instance is coreferent to the name mention in the DBWorld document. The *nearness* is measured by the number of character spaces (i.e., word distance) between two literal values (the name in the document and the affiliation information). White spaces are counted but it is not mentioned how consecutive white spaces are treated. *Text Co-occurrence* is a similar feature but is also different from *Text-Proximity* in the sense that it only requires the values to be present in the document while having no specific requirements on where they should occur or how close they should be to each other in the document. Their algorithm achieves good performance: 97.1% in precision and 79.1% in recall. However, one problem is that the authors manually and selectively picked some triples of the instances for the coreference task, e.g., name and affiliation. The features (e.g., co-occurrence) were manually identified and specifically designed for the chosen information. For domains where it is difficult to obtain the necessary domain expertise, it may not be feasible to decide what information would be important and useful; and thus it would also be difficult to identify useful features for such domains. Similar approaches were also adopted by other researchers [68, 69, 70].

Instead of trying to match free text mentions to ontology instances, other researchers developed algorithms for detecting coreferent ontology instances. These algorithms are especially useful for producing *owl:sameAs* statements to connect different Semantic Web datasets, providing the opportunity for accomplishing better data integration and query

answering in the Semantic Web. To my knowledge, the earliest such work was by Alani et al. [71]. They proposed using RDF triples of an instance and its neighborhood instances for coreference. *LogMap* [72, 73] adopts a similar approach. The basic idea is that it computes the similarity between the "labels" of two instances and picks the highest similarity between any pair of labels of the two instances as their final similarity score. Here, "label" is broadly defined but limited to objects of datatype properties. End users of *LogMap* can manually specify what datatype properties they want to use during the entity coreference process and *LogMap* will automatically extract their values before doing coreference. By default, it will only use *rdfs:label* for instance matching. The advantage of *LogMap* is that it does not require any sophisticated property matching algorithm as a pre-processing step, since it will compare the values of all pairs of specified datatype properties. However, one potential drawback would be that using the highest similarity score between values of any manually determined property pairs as the final similarity measure for two instances could result in too many false positives, because two non-coreferent instances might coincidently share highly similar values for one or more properties.

One common problem of the two approaches discussed in the above paragraph is that neither of them differentiates the importance of different triples. Without appropriately recognizing the importance of different types of triples, it would be difficult to achieve decent performance. For example, in general, person name would be expected to be more useful in disambiguating one person from others than other types of information, such as birth date, birth place, living place, etc. RiMOM [35], differently from the previous two approaches, incorporates manually specified weights for different properties. The core idea is that different properties may have quite different impact and thus for each property, a specific weight is assigned. The similarity between two instances is computed by combining such

property weights with string matching techniques. According to their evaluation results, utilizing manually assigned property weights helped to achieve 2% to 6% higher coreference F1-scores than other systems on two of three benchmark datasets for OAEI2010.

Aswani et al. [13] proposed another algorithm for matching ontology instances. Their algorithm matches person instances from an ontology converted from the British Telecommunications (BT) digital library, containing 4,429 publications and 9,065 author names. One of their focuses is to exploit the web as an external knowledge base to find information to support the coreference process. Different from the work by Mann and Yarowsky [37] where the evaluation data was web pages, here, the data is ontology instances and web is utilized as an external information source for retrieving more information. For example, they issue queries with the family name of a person instance and the title of a publication instance to search engines and see if different author instances will finally come to have the same full name. A positive answer adds confidence that the two person instances are coreferent. Some other clues include finding the publication page for different person instances, measuring the similarities between the names of person instances and the titles of their publications. The authors test their algorithm on author instances with identical family names and achieve an accuracy from 90.48% to 100% on different datasets. Similar to the paper by Hassel et al. [15], the feature set is manually identified. Also, some features require special processing. For instance, the authors manually set up some rules to determine if a returned web page is really a person's publication page or simply a page from DBLP where papers of distinct authors may co-exist.

One common feature of *LogMap* [72, 73], RiMOM [35], the algorithms by Alani et al. [71] and by Aswani et al. [13] is that they only looked at the immediate triples of an ontology instance. However, triples further away from the instance in an RDF graph may also be

helpful, particularly when there are not sufficient immediate literal triples. Furthermore, the algorithm developed by Aswani et al. and the RiMOM system both assign weights to different properties in an ontology, trying to differentiate the significance of different types of triples. However, one problem is that such property weights are manually designed. Therefore, if we want to work on a new dataset that uses a different set of predicates, additional human effort will be required to determine such weights. When there are a large number of predicates (e.g., the BTC 2012 dataset), this could be a really time-consuming process or even not feasible. In these two papers, the authors did not report any information on how much efforts/time were spent to obtain the appropriate property weights.

Combing Logical Reasoning with String Matching

In addition to adopting string matching techniques to compute the similarity between surface forms, logic based approaches were also proposed. ObjectCoref [34, 74] adopts a two-step approach for detecting coreferent instances. First, it builds an initial set of coreferent instances via reasoning, i.e., by using the formal semantics of OWL properties, such as *owl:sameAs, owl:InverseFunctionalProperty, owl:FunctionalProperty, owl:cardinality and owl:maxCardinality*. In a second step, ObjectCoref utilizes machine learning techniques to learn the discriminability of property pairs based on the coreferent instance pairs used for training. The discriminability reflects how well each pair of properties can be used to determine whether two instances are coreferent or not. Similarly, Zhishi.me [75], LN2R [76], CODI [77] and ASMOV [78] also utilize a combination of reasoning based and string similarity or lexical similarity (e.g., WordNet [79]) based techniques for matching ontology instances. One disadvantage of reasoning based approaches is that they highly depend on the correct expressions of the ontologies. For example, as reported by ASMOV researchers,

in one dataset, the *surname* property was declared to be functional while two instances with different object values of this property are said to be coreferent by the groundtruth. Another potential weakness of logic-based approaches is that they may not be applicable to non-Semantic Web data, since there are no formal semantics. For instance, for data in traditional databases and XML/CSV data, we do not have the properties listed above (e.g., owl:FunctionalProperty), thus making the logic layer to lose its power.

Ontology and Instance Matching with CrowdSourcing

The approaches discussed above are all automatic in the sense that except for having to manually specify the weights of different types of triples in some algorithms, the coreference results are achieved by an automatic system without human intervention. However, during the past a couple of years, several algorithms that consider human involvement for improving coreference results have been proposed. The whole idea behind such approaches is "The Wisdom of Crowd" where the aggregation of ideas and information by groups can often be better than those made by single individuals from groups. Typically, automatically generated results are published on some crowdsourcing platforms as evaluation tasks, such as Amazon Mechanical Turk[28] and CrowdFlower[29], and humans can then provide their judgements on the tasks to earn a little money. The feedbacks can then be used to determine the final coreference results with appropriate combination with the automatically calculated similarity or can also be utilized for active learning.

CROWDMAP [80] and ZenCrowd [81] are two such systems. Although CROWDMAP is designed for performing alignment at the ontology (schema) level, the essential crowdsourcing idea can still be applied to instance matching. However, since there are typically

[28]https://www.mturk.com/mturk/welcome
[29]http://crowdflower.com/

many more ontology instances than classes and predicates, it might be more feasible to put instance pairs of borderline matching confidence onto crowdsourcing platforms. Instances pairs that are certainly coreferent or different do not need to be judged by humans. One potential risk of utilizing crowdsourcing is that many of the evaluators are simply doing the tasks for money and are often times not spending sufficient time to really understand the tasks. This will then cause noisy results. Although some crowdsourcing services, like Amazon Mechanical Turk, try to identify featured/high-quality turkers (evaluators), researchers still need to spend lots of time filtering low-quality feedback.

2.4 Scaling Entity Coreference Systems

One of the common problems with the current approaches is scalability. Many current systems adopt exhaustive pairwise comparison between instances and they mainly focus on exploring appropriate features and metrics to compute instance similarity. However, pairwise comparison will not fit for large-scale datasets (e.g., datasets with millions of instances). Blocking is one method for subdividing mentions into mutually exclusive blocks and only mentions within the same block will be compared. In database research, one traditional method for identifying duplicate records in a database table is to scan the table and compute the value of a hash function for each record. The hash values define *buckets* to which each record will be assigned. Such hash values not only identify identical records but also records that are approximately similar [82]. In the end, in order to find duplicate records, it would be sufficient to compare only the records that fall into the same bucket. Recently, instead of finding mutually exclusive blocks, blocking is also referred to as finding a set of candidate pairs of mentions that could be coreferent [42].

Although blocking can substantially increase the speed of the comparison process in

that it only compares identifiers in the same block, there are some problems with this technique. First of all, it is not necessary that all coreferent identifiers have the same values for a single blocking property; thus, typically multiple types of information will be used in order to improve coverage on true matches. Furthermore, data quality problems need to be considered. Having noise in the data can significantly impact blocking results, causing blocking systems to place entries in the wrong buckets, and thereby preventing them from being compared to actual matching entries. In addition to only computing string similarities of the surface forms, phonetic algorithms, such as Soundex [83] and the New York State Identification and Intelligence System (NYSIIS) [84], could be adopted for better handling erroneous data. It is hoped that misspelled words will still have the same phonetic codes.

2.4.1 Blocking with Manually Identified Key

Many approaches rely on human experts to determine what information to use for blocking and are generally very effective [85, 86, 41, 87, 88, 43]. Best Five [87] is a set of manually identified rules for matching census data. Sorted Neighborhood (SN) [88] sorts all entities on one or more key values (e.g., name for a person and title for a publication) and compares identifiers in a fixed-sized window. Yan et al. [43] proposed a modified sorted neighborhood algorithm, Adaptive Sorted Neighborhood (ASN), to learn dynamically sized blocks for each record. The records are sorted based upon a manually determined key. For a record r, it automatically finds the next N records that might be coreferent to r where N could vary for different records. They claimed that changing to different keys didn't affect the results but didn't report any experimental results.

Silk [89] and Oyster [90] are two general frameworks for users to specify rules for performing record linkage, but it may be difficult for users to specify such rules for all domains.

Compared to these systems, we try to reduce the need of human input in developing entity coreference systems.

Although keys manually selected by domain experts can be very effective in many scenarios (e.g., census data), this manual process can be expensive, as the required expertise may not be available for various domains. Moreover, even when people have the necessary knowledge for identifying what information to use for blocking, they may lack the time to sit down and write down the rules.

2.4.2 Automatic Blocking Key Selection

BSL [42] adopted supervised learning to learn a blocking scheme: a disjunction of conjunctions of (method, attribute) pairs. Here, a "method" refers to how attribute values will be compared. As a concrete example, a "method" could be "computing the Jaccard similarity between two attribute values". It learns one conjunction each time to reduce as many pairs as possible; by running the learning process iteratively, more conjunctions would be obtained in order to increase coverage on true matches. However, supervised approaches require sufficient training data that may not always be available. As reported by Michelson and Knoblock [42], when 1/5 of the groundtruth was used for training, 4.68% fewer true matches were covered on the Restaurant dataset (described in Section 2.3.1). Even more important, BSL was not able to scale to a dataset with only about 23,000 records [44], since essentially it needs to try out every possible combination of (method, attribute) pairs and picks the best one (that reduces the most pairs and covers the most true matches) at each learning iteration. In order to reduce the needs of training data, Cao et. al. [8] proposed a similar algorithm that utilizes both labeled and unlabeled data for learning the blocking scheme; however the supervised nature of their method still requires a certain amount of

available groundtruth.

Differently, Adaptive Filtering (AF) [91] is unsupervised and it filters record pairs by computing their character level bigram similarity. Marlin [92] uses an unnormalized Jaccard similarity on the tokens between attributes by setting a threshold to 1, which is essentially to find an identical token between the attributes. Although it was able to cover all true matches on some datasets, it only reduced the pairs to consider by 55.35%. Considering applying this technique to large-scale datasets, this may not be a significant enough reduction to make coreference with the remaining instance pairs feasible.

2.4.3 Speeding Up Entity Coreference with Indexing Techniques

The Information Retrieval (IR) style inverted index, a technique typically used for fast data retrieval on the Web, has been widely adopted for speeding up the blocking process. An IR-based inverted index is typically built for a collection of documents where each document contains a set of terms. The index will have a term list that contains all the unique terms in this document collection; and each term will be associated with a posting list with all documents that contain this term.

PartEnum [93], BiTrieJoin [94], IndexChunk [95], FastJoin [96], All-Pairs [97], PP-Join(+) [98] and Ed-Join [99] are all inverted index based approaches. PartEnum [93] is a search based algorithm that adopts a two-level partitioning and enumeration based on Hamming distance. BiTrieJoin [94] is a trie-based method to support efficient edit similarity joins with sub-trie pruning. AllPairs [97] is a simple index based algorithm with certain optimization strategies. PPJoin+ [100] adopts a positional filtering principle that exploits the ordering of tokens in a record. Ed-Join [99] employed filtering methods that explore the locations and contents of mismatching n-grams. Similarly, IndexChunk

[95] computes asymmetric signatures on character-level n-grams as constraints for selecting candidates. Instead of performing exact matching on tokens and/or character-level n-grams, FastJoin [96] adopts fuzzy matching techniques that consider both token and character level similarity.

The Semantic Web community has also started adopting similar techniques for blocking and entity coreference. Ioannou et. al. [101] developed a system that focuses on query time duplicate instance detection on RDF data. The key technique is to index RDF resources to enable efficient look-ups. By adaptively determining the query to the index, similar instances to the query instance can be efficiently retrieved.

2.4.4 Building Scalable Systems with Feature Selection

Most of the current research focuses on how to reduce the total number of pairwise comparisons between entity mentions; however, it is also important to speed up a single pairwise comparison. This can be generalized as a pruning process where we prune the less important parts of an instance's context information. For example, when we compare the similarity between a pair of instances, the system predicts if the remaining unconsidered context will overturn the current decision made based upon what has already been compared. If it is not worth continuing to explore the rest of the context, the system will simply stop and continue with the next pair of instances. This is similar to the feature selection problem [102] where algorithms are developed to select the right features for a specific problem in order to reduce computational complexity. The key problem here is how to stop at the right places (selecting the right features). Stopping too soon may cause the system to lose some number of true matches; while going too far could potentially bring in unnecessary computational costs.

Chapter 3

A Domain-Independent Entity Coreference Resolver for Linking Ontology Instances

We have developed and published a domain-independent entity coreference resolver for detecting *owl:sameAs* links between ontology instances [16, 103]. I will describe this work in the remainder of this chapter, formally presenting the algorithm and evaluation results.

Entity coreference in the Semantic Web can be modeled as following. Given an RDF graph G and a set of ontology instances I, we find all the coreferent instance pairs $EQ(G, I)$ as represented in Equation 3.1:

$$EQ(G, I, t) = \{(e_i, e_j) | (e_i, e_j) \in I \times I \wedge InstanceSim(C(G, e_i), C(G, e_j)) > t\} \quad (3.1)$$

Where e_i and e_j represent two ontology instances; the function $C(G, e_i)$ extracts the context information for instance e_i in the given RDF graph G; $InstanceSim$ is a function that

computes the similarity score between the context information of two ontology instances and t is a threshold on this similarity score. An entity coreference algorithm computes the similarity scores between instance pairs in a given dataset and pairs whose similarity score is higher than the given threshold t are treated as coreferent.

In order to find $EQ(G, I, t)$, several steps have been employed and I will present the details of each step in the rest of this chapter:

- We firstly find the context information of an ontology instance in a given RDF graph;
- Next, we automatically assign appropriate weights to different parts of the context;
- Finally, we develop an algorithm that could appropriately utilize such weighted context information to compute the similarity score between instance pairs.

3.1 Selecting a Neighborhood RDF Graph

In our accomplished work [16, 103], we use the RDF graph as the sole source of context information. We collect paths in an RDF graph within a certain distance to an instance (the root node) that we do coreference on. We define a path as a sequence of nodes and predicates in an expansion chain in Equation 3.2:

$$path =< i, predicate[1], node[1], predicate[2], node[2], ..., predicate[n], node[n] > \quad (3.2)$$

where i is the instance, $node[i]$ $(i > 0)$ is any other expanded node from the RDF graph and $predicate[i]$ is a predicate that connects two nodes in a path. We define a function $depth(path)$ that counts the number of predicates in a path. We will use the operator $+$ for tuple concatenation, e.g., $< a, b, c > + < c, d, e >=< a, b, c, d, e >$.

Algorithm 1 Neighborhood(G, i), i is an ontology instance and G is an RDF graph; returns a set of paths for i collected from G

1. $P \leftarrow$ *all predicates in* G
2. $path \leftarrow < i >$
3. $expansion\_set \leftarrow \{path\},\ paths \leftarrow \emptyset$
4. **for all** $path' \in expansion\_set$ **do**
5. $\quad last \leftarrow$ *last node in* $path'$
6. $\quad$ **if** *last is literal* **or** (*last is URI* **and** $depth(path') = depth\_limit$) **then**
7. $\quad\quad paths \leftarrow paths \bigcup path'$
8. $\quad$ **else if** $depth(path') < depth\_limit$ **then**
9. $\quad\quad /* Expand\ from\ subject\ to\ object */$
10. $\quad\quad triples \leftarrow \bigcup_{p \in P} \{t | t =< last, p, o > \wedge\ t \in G\}$
11. $\quad\quad$ **for** $t =< s, p, o > \in triples$ **do**
12. $\quad\quad\quad path\_new \leftarrow path' + < p,\ o >$
13. $\quad\quad\quad expansion\_set \leftarrow expansion\_set \bigcup path\_new$
14. $\quad\quad /* Expand\ from\ object\ to\ subject */$
15. $\quad\quad triples \leftarrow \bigcup_{p \in P} \{t | t =< s, p, last > \wedge\ t \in G\}$
16. $\quad\quad$ **for** $t =< s, p, o > \in triples$ **do**
17. $\quad\quad\quad path\_new \leftarrow path' + < p^{-},\ s >$
18. $\quad\quad\quad expansion\_set \leftarrow expansion\_set \bigcup path\_new$
19. $\quad expansion\_set \leftarrow expansion\_set - path'$
20. **return** *paths*

Algorithm 1 formally presents this expansion process. Starting from an instance, we search for triples whose subject or object equals to the URI of this instance and record those expanded triples. With the expanded triples, if the objects or subjects are still URIs or blank nodes, we repeat this search or expansion process on them to get further expanded triples until we reach a depth limit or a literal value, whichever comes first. In an RDF graph, a blank node (or anonymous resource) is a node that is neither identified by a URI nor is a literal. Blank nodes have a node ID which is limited in scope to a serialization of a particular graph, i.e., the node *node*[1] in one RDF graph does not represent the same node as a node named *node*[1] in any other graph. At line 17, we use p^{-} to denote a predicate p when expanding from object to subject.

We implemented this expansion process as breadth-first search. In order to control the number of paths generated, we set a depth limit (the maximum number of predicates in

a path) of 2. With this limit, we've discovered that it is sufficient to get enough context information. For example, in the RKB dataset[1], given a person instance, we can find its name, affiliation, and the URIs of this person's publication instances at depth 1; going further to depth 2, we will have the titles and dates of this person's publications, the URIs of the coauthors of these publications, etc.

With our expansion process, we end up having a set of paths for each ontology instance, starting from that instance and ending with a URI or a literal value. When ending on a blank node, we do not record that path because we cannot simply compare two blank nodes and see if they are identical. However, we rely on paths that go through blank nodes to get further literals and URIs before the stopping criteria is met. As illustrated in Figure 3.1, starting from the root (an instance), we get to node 1, 2 and 3 by searching triples that use the root node as subject or object; then we reach node 4, 5, 6 and 7 by further expanding node 2, so on and so forth. We will explain P and F from Figure 3.1 in Section 3.2.

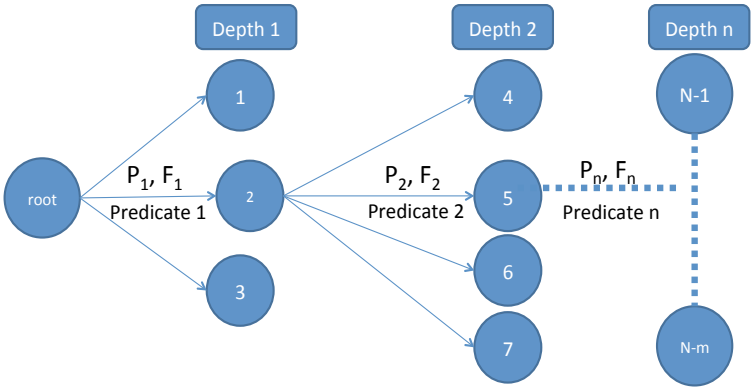


Figure 3.1: Expansion Result

[1]This is one of the datasets that we use for evaluation in Section 3.4.

3.2 Calculating Path Weights

In this section, we will present our approach for learning the discriminability of RDF triples, explaining how we could utilize the collected context information appropriately. Generally, each triple has its own importance, reflecting its possible level of discrimination to the ontology instance from which it originally comes from[2]. Our discriminability learning approach is domain-independent. Given a specific dataset, without a domain-independent and automatic discriminability learning algorithm, we need to manually determine the importance of each triple. When disambiguating person instances, we need to manually determine that person name can be more discriminative than birthplace or hometown, and others like such. Our approach, given a new dataset, takes the entire dataset (triples) as input and automatically computes the discriminabilities regardless of the domain of that dataset, such as academia domain or some others.

3.2.1 Predicate Discriminability

Thinking broadly, we can measure the discriminability of a triple by only looking at what its predicate is. As for a predicate, the more diverse value set it has, the more discriminating it will be. Triples with different predicates, such as *has_publication_date* and *has_author*, could have different discriminabilities. Equations 3.3 and 3.4 show how we compute predicate discriminability:

$$Per_{p_i} = \frac{|set\ of\ distinct\ objects\ of\ p_i|}{|set\ of\ triples\ that\ use\ p_i|} \tag{3.3}$$

where Per_{p_i} represents a percentage value for predicate p_i, which is the size of p_i's distinct object value set divided by its number of occurrences in the entire dataset. We record the

[2]This is related to finding the neighborhood graph for an instance as introduced in Section 3.1.

largest percentage value over all predicates as Per_{max}, and then normalize such values so that the most discriminating predicate has a discriminability of 1. The normalization is shown in equation 3.4:

$$P_{p_i} = \frac{Per_{p_i}}{Per_{max}} \tag{3.4}$$

where P_{p_i} is the predicate discriminability for predicate p_i.

Depending on what category of instances is being compared, a predicate may be used in the subject-to-object direction or reversely. A predicate that discriminates well in one direction may not do well in the other. This is related to how we expand an instance to collect neighborhood triples in Section 3.1. Basically, when we do the expansion, we use a URI both as the subject and the object, so a predicate has different discriminabilities to the two directions.

To clearly represent discriminabilities, for a given predicate p_i, we use Per_{p_i} and $Per^-_{p_i}$ to denote the percentage values to the object and subject direction respectively; then the predicate discriminabilities to the two directions are denoted as P_{p_i} and $P^-_{p_i}$ respectively. Equations 3.3 and 3.4 compute predicate discriminabilities to the object direction. The discriminabilities to the subject direction can be computed in the same manner by replacing appropriate variables.

Here, we show how to calculate our predicate discriminability with two concrete examples from the RKB dataset. In total, 6,313,274 triples use the predicate *has_author* (with domain of publication class and range of person class); among these triples, there are 3,986,181 distinct object values and 2,515,439 distinct subject values. From Equation 3.3, we have:

$$Per_{has\_author} = \frac{3,986,181}{6,313,274} = 0.63,\ Per^-_{has\_author} = \frac{2,515,439}{6,313,274} = 0.398 \tag{3.5}$$

Because the maximum percentage values to both directions (Per_{max} and Per^{-}_{max}) are both 1 in this dataset, based on Equation 3.4, we have:

$$P_{has\_author} = \frac{Per_{has\_author}}{Per_{max}} = \frac{0.63}{1} = 0.63, \tag{3.6}$$

$$P^{-}_{has\_author} = \frac{Per^{-}_{has\_author}}{Per^{-}_{max}} = \frac{0.398}{1} = 0.398 \tag{3.7}$$

Therefore, when disambiguating between publication instances, having a common author can be more discriminative than having a common publication when doing coreference on person instances. Another example is the *has_publication_year* predicate (a datatype property). 2,973 triples use this predicate with 152 distinct object values. So, we have:

$$Per_{has\_publication\_year} = \frac{152}{2,973} = 0.05 \tag{3.8}$$

Based on Equation 3.4, we have:

$$P_{has\_publication\_year} = \frac{Per_{has\_pub\_year}}{Per_{max}} = \frac{0.05}{1} = 0.05 \tag{3.9}$$

The intuition behind our predicate discriminability is that the discriminability of a triple is determined by its predicate. And such discriminability will then contribute to the entity coreference process. For example, if two publications happen to have the triples with the same object value via predicate *has_publication_year* that only has a weight of 0.05, then such a coincidence does not really add much value to determine if they are coreferent; however, predicate *has_author* shows a much higher discriminability to the object direction (0.63), so that having equivalent object values for this predicate (having the same author) will give a better idea that these two publications be coreferent.

3.2.2 Weighted Neighborhood Graph

With triple discriminability and the context information, we assign each path in the neighborhood graph a weight, indicating its importance to the root node. The weights combine two elements, the learned discriminability and a discount value.

As previously shown in Figure 3.1, P_1 and P_2 represent the discriminabilities for the two triples ending on node 2 and 5 respectively. We also add another parameter, called factor, to each node, indicating how important a node is to its parent. For example, in Figure 3.1, F_1 is the factor of node 2 to the root node and its value is 1/3 because three triples get expanded from the root. Each of the three nodes (node 1, 2 and 3) only represents one-third of their parent conceptually. The underlying semantics of this factor is to portion out the importance of one node to its expanded nodes in an equal manner.

With the factors and triple discriminability, we adopt a distance based discounting approach to assign weights to paths in the neighborhood graph, with Equation 3.10:

$$W_{path} = \prod_{i=1}^{depth(path)} P_i * F_i \tag{3.10}$$

where the function *depth* counts the number of predicates in a path; P_i and F_i represent the discriminability and factor for each triple in the path respectively. The intuition here is that as we expand further in the RDF graph, more noisy data could be introduced. In order not to overwhelm the context, the discriminability of each expanded triple should be appropriately adjusted. We call this a distance-based discounting approach.

3.2.3 Predicate Discriminability Overestimation

Currently, when counting the size of the distinct object/subject value sets, we assume that if any two objects/subjects are syntactically distinct, then they truly represent different things. However, they could actually represent the same real world entity. With such unknown coreferent objects/subjects, we are actually overestimating the discriminability. But if we assume that for every predicate such unknown coreferent relationships occur uniformly throughout the dataset, we actually overestimate all predicates by the same proportion. Thus our current approach still gives reasonable discriminability.

Figure 3.2 gives an example of this situation. We have two predicates: PredicateA (P_A)

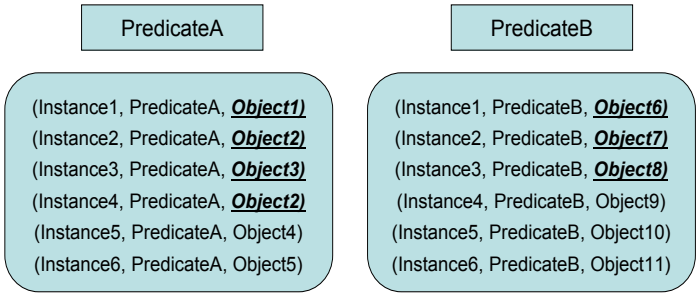


Figure 3.2: Predicate Discriminability Overestimation

and PredicateB (P_B) from the same dataset; each of them is used by six triples as listed in the two boxes respectively. Assuming Per_{max} is 1 for this dataset, based on Equations 3.3 and 3.4, to the object direction, we can calculate their discriminability:

$$P_{P_A} = \frac{Per_{P_A}}{Per_{max}} = \frac{5}{6} = 0.83,\ P_{P_B} = \frac{Per_{P_B}}{Per_{max}} = \frac{6}{6} = 1 \tag{3.11}$$

Now, if we assume the underlined objects are actually coreferent, the predicate discriminability of these two predicates will change to:

$$P_{P_A} = \frac{Per_{P_A}}{Per_{max}} = \frac{3}{6} = 0.5,\ P_{P_B} = \frac{Per_{P_B}}{Per_{max}} = \frac{4}{6} = 0.67 \tag{3.12}$$

In this case, we are overestimating the predicate discriminability for both predicates due to unknown coreferent instances. In our unsupervised method for learning predicate discriminability, it is difficult to consider such unknown coreferent information; therefore, we make the assumption that we overestimate the discriminability for every predicate by about the same proportion. From our current experiments (to be presented in Section 3.4), adopting our proposed predicate discriminability does significantly improve system performance compared to comparison systems (to be introduced in Section 3.4.3) that don't use it.

3.2.4 Missing Value

Currently, we don't consider missing values when calculating our predicate discriminability values. A version of Equation 3.3 that takes missing values into account is shown in Equation 3.13:

$$Per'_{p_i} = \frac{|set\ of\ distinct\ objects\ of\ p_i| + 1}{|triples\ that\ use\ p_i| + |instances\ that\ should\ use\ p_i\ but\ don't|} \tag{3.13}$$

where Per'_{p_i} represents the percentage value for predicate p_i.

Compared to Equation 3.3, the additional "1" in the numerator represents *null value* and the second part ($Missing = |instances\ that\ should\ use\ p_i\ but\ don't|$) in the denominator is the number of instances that should have a value for predicate p_i but actually don't. The

problem here is how to determine $Missing$. In an ontology, a predicate may not have its domain declared thus making it difficult to calculate $Missing$. Another option is to compute predicate discriminability with respect to each individual class as shown in Equation 3.14:

$$Per'_{(C,p_i)} = \frac{|set\ of\ distinct\ objects\ of\ p_i| + 1}{|triples\ that\ use\ p_i| + |instances\ of\ C\ that\ don't\ use\ p_i|} \tag{3.14}$$

where C is an ontology class. With this option, we compute the discriminability of all predicates for each individual class; thus a predicate may have different discriminabilities when paired with different classes. As shown in Figure 3.1, during expansion, we need to determine the discriminability of an edge (a predicate). However, the nodes to expand (e.g., the root node in Figure 3.1) may not have class types declared or may have multiple types; therefore one problem here is that we may not be able to decide which discriminability to use during the expansion process. For instance, in the RKB dataset, a person instance is also declared to be an *owl:Thing*, *Legal-Agent* and *Generic-Agent*. For predicate p_i, although we could compute its discriminability by Equation 3.14 with respect to each of these classes, it is difficult to decide which one to use when seeing an edge of p_i. In our current work, we didn't implement these two alternate options due to the problems discussed above when applying them generally to Semantic Web data. For future work, we will explore approaches for calculating predicate discriminability by appropriately taking into account missing values.

3.3 Exhaustive Pairwise Entity Coreference based upon Weighted Neighborhood Graph

3.3.1 Algorithm Design

Algorithm 2 presents the pseudo code of our entity coreference algorithm for ontology instances. In this description, x and y are two ontology instances; $N(G, x)$ returns the context (a set of weighted paths) of instance x in RDF graph G; $E(path)$ returns the end node of a path; the function $PathComparable$ indicates if two paths are comparable; Sim is a string matching algorithm that computes the similarity score between two literals.

Algorithm 2 Compare(N_x, N_y), N_x is the context N(G,x) and N_y is N(G,y); returns a float number (the similarity of x and y)

1. $total\_score \leftarrow 0, total\_weight \leftarrow 0$
2. **for all** *paths* $m \in N_x$ **do**
3. **if** $\exists$*path* $n \in N_y$, $PathComparable(m, n)$ **then**
4. $path\_score \leftarrow 0, path\_weight \leftarrow 0$
5. **if** $E(m)$ *is literal* **then**
6. $path\_score \leftarrow \max_{n' \in N_y, PathComparable(m,n')} Sim(E(m), E(n'))$
7. /* path n' has the highest score with m */
8. $path\_weight \leftarrow (W_m + W_{n'})/2$
9. **else if** $E(m)$ *is URI* **then**
10. **if** $\exists path\ n' \in N_b, PathComparable(m, n'), E(m) = E(n')$ **then**
11. $path\_score \leftarrow 1$
12. /* path n' has identical end node with m */
13. $path\_weight \leftarrow (W_m + W_{n'})/2$
14. $total\_score \leftarrow total\_score + path\_score * path\_weight$
15. $total\_weight \leftarrow total\_weight + path\_weight$
16. **return** $\frac{total\_score}{total\_weight}$

The essential idea of our entity coreference algorithm is that we adopt the *bag-of-paths* approach to compare paths between ontology instances. Information retrieval systems often treat a document as a *bag-of-words* [104] where the text is treated as an unordered collection of words. Analogously, we treat an instance as a *bag-of-paths*. Through the expansion process (discussed in Section 3.1), for an instance, we find a collection of paths for it

without considering the ordering of the paths.

As presented in Algorithm 2 and illustrated in Figure 3.3, for each path (m) of instance x, we compare its last node to that of every comparable path of instance y and choose the highest similarity score, denoted as *path_score*. Also, we need to determine the weight of this path score. Here, when considering the weight, we take into account both the weight (W_m) of path m and the weight ($W_{n'}$) of the path n' of instance y that has the highest similarity to path m. Then we use the average of W_m and $W_{n'}$ as the path weight for path m. We then repeat the process for every path of instance x. With the pairs of (path score, path weight) for a pair of instances, we calculate their weighted average in order to have the final similarity score between the two instances. The same process is repeated for all pairs of ontology instances of comparable categories, i.e., person-to-person and publication-to-publication. At line 16, the similarity score (a floating number) for a pair of instances is returned.

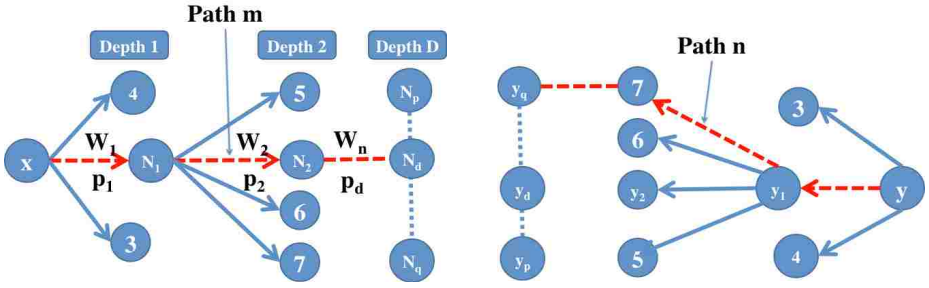


Figure 3.3: An Example of Computing Comparable Paths between Two Weighted Neighborhood Graphs

3.3.2 Path Comparability

As described, we compare the last nodes of instance a's paths to those of the comparable paths of instance b. One question here is how to determine if two paths are comparable. For example, the following two paths are not comparable (predicates in bold and italic):

- (personA, ***attend_event***, eventA, ***has_event_date***, "2010-06-01");
- (personA, ***has_publication***, article1, ***has_publication_date***, "2010-06-01").

Although the two last nodes are all date information and thus are comparable, they actually come from paths with different underlying semantics.

In our current approach, two paths are comparable if they satisfy the condition shown in Equation 3.15:

$$\begin{aligned} &PathComparable(path_1, path_2) = true \equiv \\ &(depth(path_1) = depth(path_2)) \ \wedge \\ &(\forall i \in [1, depth(path_1)], \\ &PredicateComparable(path_1.predicate[i], path_2.predicate[i]) = true) \end{aligned} \tag{3.15}$$

where *PathComparable* and *PredicateComparable* are two functions, indicating if two paths or two predicates are comparable; *depth* counts the number of predicates in a path; $path_1.predicate[i]$ and $path_2.predicate[i]$ return the ith predicate in two paths respectively.

There are several questions coming out from Equation 3.15. The first one is how to determine the comparability of two predicates, which will help to determine path comparability. In some situations, the comparability of predicates is not very clear. For instance, the two predicates *author_on* and *edit_on* can be vague in their comparability. For a publication, a person that did some edits on it does not necessarily have to be listed as an author of it. In such a circumstance, without letting such two predicates be comparable, we might miss some true matches while adding them in might hurt precision.

We say two predicates are comparable if the knowledge base (KB) entails that one is the subproperty of another (obviously, this means equivalent properties are also comparable). For our experiments, we created these mapping axioms manually. For example, the two predicates are comparable: *citeseer:fullname* and *foaf:name*[3]. They are from different ontologies, but both represent person name. Ontology alignment [105, 106, 107], a well studied topic in the Semantic Web, can help automatically determine predicate and class comparability across multiple ontologies, which is out of the scope of this dissertation. In Chapter 7, we will present a value-based approach for automatically determining predicate comparability.

Furthermore, when determining the comparability of two paths, we only care about the predicates in the paths but to ignore the intermediate nodes. The reason is that in two paths, intermediate nodes are URIs and distinct URIs do not necessarily represent distinct real world entities. Distinct URIs can actually represent two coreferent instances. Since we do not have the complete coreference knowledge between URIs, it is hard to involve intermediate nodes when determining path comparability. One possible solution is that we adopt an iterative entity coreference algorithm on all instances of comparable classes in a dataset and then take the coreference results of the current iteration into account in further iterations to help better determine path comparability. However, this will require certain level of system scalability because a dataset can have millions of instances.

3.3.3 Node Similarity

Given that two paths are comparable, if the two last nodes are two literal values, e.g., person names or publication titles, then we adopt the JaroWinklerTFIDF [108] string matching

[3]FOAF stands for Friend of a Friend. More details are available at http://www.foaf-project.org/.

algorithm to compute their similarity unless otherwise specified. In another situation, if the two nodes are both URIs in the RDF graph, then we will simply check if they are identical. The similarity score between two literals ranges from 0 to 1 while the score between any pair of URIs will be either 0 (not match) or 1 (identical). If the URIs do not match, the similarity is computed by further expanding those URIs as we presented in Section 3.1.

3.3.4 Comparing All Nodes in Paths

One design choice we made is to compute the *path_score* between two comparable paths by only comparing their end nodes. Another option is to take into account those intermediate nodes when calculating *path_score* as shown in Algorithm 3.

Algorithm 3 Compare_All_Nodes(N_a, N_b), N_a and N_b are the context of instances a and b collected with Algorithm 1; returns the similarity between a and b

```
1. total_score ← 0, total_weight ← 0
2. for all paths m∈N_a do
3.   if ∃path n∈N_b, PathComparable(m, n) then
4.     path_weight ← 0, path_score ← 0
5.     for all paths n'∈{p|p ∈ N_b ∧ PathComparable(p, m)} do
6.       score ← 0, count ← 0
7.       for i <= depth(m) do
8.         if m.node[i] is literal and n'.node[i] is literal then
9.           score ← score + Sim(m.node[i], n'.node[i])
10.          count ← count + 1
11.        else if m.node[i] is URI and n'.node[i] is URI then
12.          if m.node[i] = n'.node[i] then
13.            score ← score + 1
14.          count ← count + 1
15.      if score/count > path_score then
16.        path_score ← score/count
17.    n' ← path with the highest score compared to m
18.    if m.node[last] is literal then
19.      path_weight ← (W_m + W_n')/2
20.    else if m.node[last] is URI and path_score ≠ 0 then
21.      /* path n' has at least one identical intermediate URI node with m */
22.      path_weight ← (W_m + W_n')/2
23.    total_score← total_score + path_score * path_weight
24.    total_weight ← total_weight + path_weight
25. return total_score/total_weight
```

From line 5 to 16, we calculate the *path_score* between two comparable paths as the average of all matching scores between nodes at corresponding positions; and we pick the maximum *path_score* between path m of instance a and all its comparable paths of instance b. Note, intermediate nodes must either be URIs or blank nodes. Blank nodes have a node ID which is limited in scope to a serialization of a particular graph, i.e., a blank node named *nodeA* in one RDF graph does not represent the same node as a node with the same name in any other graph. Therefore, we do not compare two blank nodes or a blank node and a URI but only compare two URIs or two literals from line 8 to 14. At line 20, if two paths whose nodes are all URIs or blank nodes have a *path_score* of 0, meaning none of their nodes match, we do not update *path_weight*. Because *path_weight* is initialized to be zero at line 4, *total_score* and *total_weight* will not be updated either at line 23 and 24. In this case, we are not applying any penalty to a path that ends on a URI and doesn't match any of its comparable paths from the other instance.

The rationale to only consider end nodes (as presented in Algorithm 2) is that the intermediate nodes from two paths could be syntactically different but are actually coreferent instances. So, if we apply penalties to mismatching intermediate nodes, it is possible for the system to miss some true matches. One potential advantage of matching intermediate nodes is that it may improve the system's precision because it is possible that the middle nodes are indeed different instances but the end nodes are coincidentally the same. We hypothesize that the improvement on precision cannot compensate the sacrificed recall when considering intermediate nodes and we will experimentally compare Algorithm 2 and Algorithm 3 in Section 3.4.4.

3.3.5 The Open World Problem

Another challenge that we face is that we cannot make a closed-world assumption, instead we need to deal with open-world [109]. We cannot assume something we don't know is false, everything we don't know is undefined. Within a Semantic Web dataset, some information can be missing. Our RKB dataset is composed of several subsets of the complete RKB dataset, such as ACM, IEEE, DBLP, CiteSeer, etc. Each of them, in the case of person instances, only includes a certain portion of their information. For instance, these datasets might only contain some of these person instances' publications.

In our entity coreference algorithm, we try to relieve this Open World problem. First of all, we do not apply penalties to mismatches on URIs. As shown in Algorithm 2, if the last node of path m of instance a is a URI but it doesn't match any last node of comparable paths of instance b, we do not add any weight to *total_weight*. These mismatched URIs are expanded to get further literals and other URIs to determine the similarity.

Second, we do not apply any penalties on missing information. If there are no paths of instance b that are comparable to path m of instance a, we do not apply any penalties. The intuition behind our approach is that we compare every path present in the context and apply appropriate penalties; in the meanwhile, mismatches that are potentially caused by information incompleteness cannot simply be treated as real mismatches. We would like to investigate more sophisticated solutions to this problem in future work.

3.4 Evaluation Results

In this section, we will evaluate the proposed *EPWNG* algorithm for detecting coreferent ontology instances. We will first introduce our evaluation datasets and metrics and then

present our evaluation results.

3.4.1 Evaluation Datasets and Metrics

Datasets. We evaluate our entity coreference algorithm on two RDF datasets: RKB[4] [17] and SWAT[5]. For RKB, we use eight subsets of it: ACM, DBLP, CiteSeer, EPrints, IEEE, LAAS-CNRS, Newcastle and ECS. This collected dataset has 82 million triples (duplicates are removed), 3,986,676 person instances and 2,664,788 publication instances. The SWAT dataset consists of RDF data parsed from the downloaded XML files of CiteSeer and DBLP. The converted RDF files are then parsed and stored into database, resulting in a total of 26 million triples. Within our SWAT dataset, there are 904,211 person instances and 1,532,758 publication instances.

Although the two datasets share some information, the main difference is that they use different ontologies, so that different predicates are involved. Their coverage of publications could also be different. Additionally, some information may be ignored from the original XML files for the SWAT dataset during transformation. Note that all *owl:sameAs* statements in both datasets are ignored while we collect the context for instances as described in Section 3.1. They are only used for evaluating our results but not for facilitating our entity coreference process in any sense.

We evaluate on four instance categories: RKB Person, RKB Publication, SWAT Person and SWAT Publication. The groundtruth was provided as *owl:sameAs* statements that can be crawled from the RKB CRS service and downloaded from the SWAT website as an RDF dump respectively. Due to scalability issues, we randomly picked 1,579 RKB Person, 2102 RKB Publication, 1010 SWAT Person, and 1,378 SWAT Publication instances.

[4]http://www.rkbexplorer.com/data/
[5]http://swat.cse.lehigh.edu/resources/data/

Metrics. Our algorithm does entity coreference on every pair of instances in the test sets and stores results in the form of (instanceA, instanceB, score). In our evaluations, we use the standard measures: precision, recall and F1-score as computed in Equations 3.16 and 3.18:

$$Precision_t = \frac{|correctly\ detected\ pairs|}{|totally\ detected\ pairs|}, \tag{3.16}$$

$$Recall_t = \frac{|correctly\ detected\ pairs|}{|true\ matches\ in\ test\ set|} \tag{3.17}$$

$$F1\text{-}Score_t = 2 * \frac{Precision_t * Recall_t}{Precision_t + Recall_t} \tag{3.18}$$

where t represents threshold in all three equations.

There are a few things to note about our evaluations. First of all, in our current work, we evaluate our algorithm on different instance categories from two different datasets. Groundtruth of the RKB dataset can be downloaded from their website. To verify the soundness of the RKB groundtruth, we manually verified 300 coreferent pairs of person instances and publication instances respectively, while there are 81,556 and 148,409 in total for person and publication respectively in our collected RKB dataset. For the SWAT dataset, we manually labeled the groundtruth.

Furthermore, because the *owl:sameAs* predicate is transitive, i.e., if A is coreferent with B which is also coreferent with C, we will also have A and C are coreferent due to transitivity. In order to give the best correct evaluation results, we materialized all the coreferent pairs that can be achieved through reasoning on transitivity. Please note that we do not materialize reflexivity and symmetry.

Finally, in order to guarantee the completeness of the RKB groundtruth, we adopted a

lazy or passive approach. We run our entity coreference algorithm on the two RKB test sets, and apply thresholds from 0.3 to 0.9 to evaluate the results based upon the groundtruth provided by RKB. Then we pick the comparison system (to be formally presented in section 3.4.2) that obtains the lowest precision, find out all the pairs that are detected by this system but are said to be not coreferent according to the groundtruth. For these *wrongly* detected pairs, we manually check each of them to see if any pair should be coreferent. We rely on the authors' DBLP and real homepages to perform such checks. Through this lazy-verification step, we were able to find 295 missing coreferent pairs for the RKB person test set. Mostly, RKB misses a coreferent pair of person instances because different subsets of RKB share little common publications for the two instances.

3.4.2 Comparison Systems

In order to show the effectiveness of our proposed entity coreference algorithm, we compare our algorithm to comparison systems that are not equipped with all the features we have presented. In our proposed system, we have the following features: expansion (E#) (# represents the depth of expansion), discriminability (P, representing predicate based triple discriminability), discount (D) which is implemented by using the factor. For example, the comparison system E2-D says it expands to depth 2, doesn't use triple discriminability but uses the discount of each expanded triple. So, for E2-D, Equation 3.10 will change to Equation 3.19:

$$W_{path} = \prod_{i=1}^{depth(path)} F_i \tag{3.19}$$

where path depth is 2. For system E2-P, it uses predicate discriminability in the way that it propagates the discriminabilities of the triples along the expansion chain. Although such propagations can be viewed as one type of discount, our real discounting is from the factors.

So, for this system, the weight of a path is computed with Equation 3.20:

$$W_{path} = \prod_{i=1}^{depth(path)} P_i \tag{3.20}$$

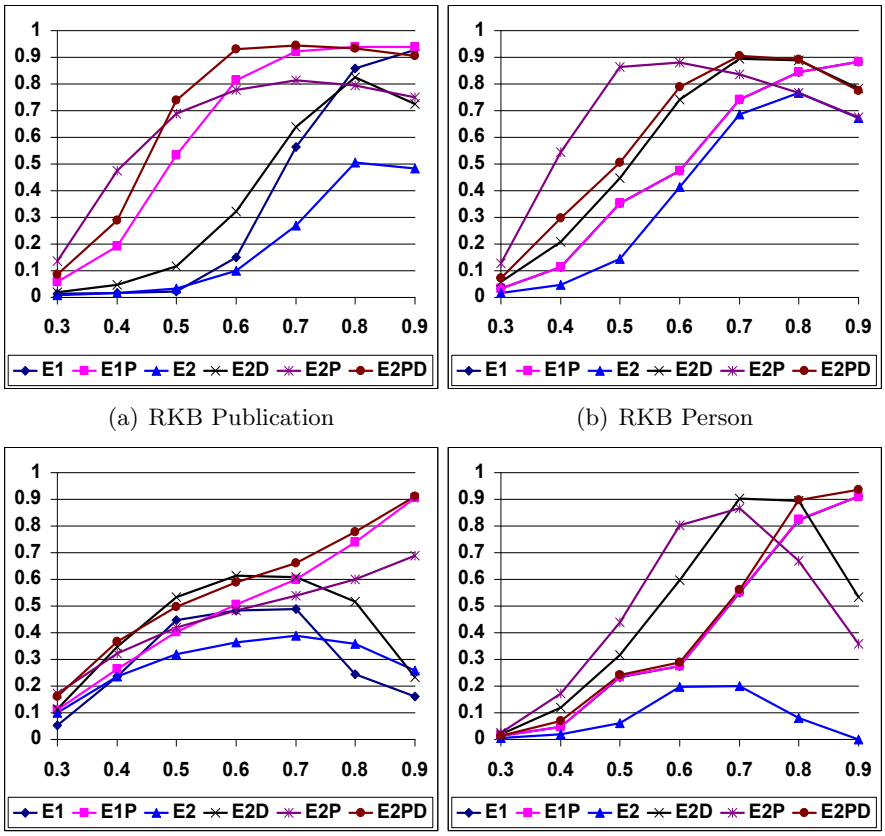
where P_i is the predicate discriminability of a triple at depth i. For systems E1 and E2, the weight for every path in the neighborhood graph is set to 1. Our proposed algorithm, E2-P-D, then uses depth 2 expansion and adopts discounts and predicate based triple discriminability to form path weight.

3.4.3 Evaluating Different Context Weighting Schemes

Figures 3.4(a) to 3.4(d) show the F1-scores of our algorithm on the RKB publication, RKB person, SWAT publication and SWAT person datasets respectively. The x-axes are thresholds and the y-axes are F1-scores. In this experiment, we adopted the JaroWinklerTFIDF string matching algorithm developed by Cohen et al. [108].

From the F1-scores, we can see that our distance-based discounting entity coreference algorithm, E2-P-D, achieves the best performance on all four datasets. On SWAT Person, although E2-P-D doesn't dominate the other systems for all thresholds, its best performance is higher than that of the other comparison systems. The F1-scores that our algorithm achieves for RKB publication, RKB person, SWAT publication and SWAT person are 94.4%, 90.6%, 91.0% and 93.8% at threshold 0.7, 0.7, 0.9 and 0.9 respectively.

System E1 is our baseline system in the sense that there is no discriminability included, no discounts at all and that it only considers adjacent triples. Compared to E1, E2 finds neighborhood graphs in a broader range. But without discounts and discriminability, it is clearly worse than E1 for RKB publication and RKB person and SWAT person (for RKB Person, the E1 and E1-P curves are nearly overlapping and are both dominating the E2



(a) RKB Publication (b) RKB Person

(c) SWAT Publication (d) SWAT Person

Figure 3.4: F1-scores for RKB Publication, RKB Person, SWAT Publication and SWAT Person. In each subfigure, the x-axis is the threshold and the y-axis is the F-score.

curve); for SWAT publication, although E2 has better results at thresholds 0.8 and 0.9, its best F1-score is much worse than that of E1. Such comparison shows that broader contexts can actually have negative impact if not appropriately managed.

By comparing E1-P to E1, it is not very clear that if only adding triple discriminability on adjacent triples gives better results. For the two publication datasets, adding such discriminability did lead to better results (except for threshold 0.9 and 0.5 for RKB publication and SWAT publication respectively); however, for the other two datasets, these two systems achieve very similar performance. Note that sometimes the curves of the two systems on the two person datasets are not very clear because they are overlapping.

In general, E2-P is better than E2 for all datasets, though they achieve similar results for RKB person at thresholds 0.8 and 0.9. Different from the comparison between E1 and E1-P, adding triple discriminability to broader contexts significantly improved system performance. This shows the effectiveness of using a broader context with better management. The differences between E2 and E2-D show that by only applying factor discounts can also give us a significant improvement on all datasets. This verifies the effectiveness of discounting. For SWAT publication, although E2-D and E2 have similar results at thresholds 0.3 and 0.9, the best F1-score of E2-D is much higher than that of E2.

Finally, our proposed E2-P-D algorithm is able to achieve the best F1-score for all datasets. Although E2-P-D is not as good as some other comparison systems at low thresholds, its performance improves as threshold rises, often topping out higher than others in the study. E2-P-D dominates E2-D on RKB publication at all thresholds and on RKB person except at threshold 0.9; also, on SWAT publication, E2-P-D is clearly better than E2-D from thresholds 0.7 to 0.9. Again, this verifies the effectiveness of using predicate discriminability. E2-P-D shows significant improvement over E2-P for the two publication datasets (thresholds 0.5 to 0.9 for RKB publication and thresholds 0.4 to 0.9 for SWAT publication); for the two person datasets, it is not significantly better than E2-P but is able to be on top at high thresholds. Such results demonstrate the effectiveness of combining predicate discriminability and the discounting factor.

3.4.4 End Nodes Only vs. All Nodes in Paths

In Sections 3.3.1 and 3.3.4, we discussed two alternatives of computing the similarity between two comparable paths: only matching end nodes or matching all nodes of two paths. In this section, we experimentally compare these two alternatives on our four test sets as

Table 3.1: Matching End Nodes vs. Matching All Nodes. We bold the higher scores that a system achieves than the other for each threshold on a dataset and also underline the best F1-scores for all thresholds for each dataset.

Dataset	Metric	System	Threshold				
			0.5	0.6	0.7	0.8	0.9
RKB Publication	Precision	End-Node	58.75	87.74	95.42	97.71	**99.71**
		All-Node	**64.46**	**89.68**	**96.41**	**98.44**	99.41
	Recall	End-Node	**100**	**99.28**	**93.42**	**89.29**	**82.72**
		All-Node	99.58	95.63	88.28	68.06	30.02
	F1-score	End-Node	74.02	**93.15**	**<u>94.41</u>**	**93.31**	**90.42**
		All-Node	**78.26**	92.56	92.16	80.48	46.12
RKB Person	Precision	End-Node	34.09	67.18	88.40	97.89	**99.21**
		All-Node	**43.18**	**75.76**	**89.39**	**98.06**	97.31
	Recall	End-Node	**98.32**	**95.88**	**92.94**	**81.92**	**63.75**
		All-Node	96.38	91.76	78.64	59.46	21.28
	F1-score	End-Node	50.63	79.00	**<u>90.61</u>**	**89.19**	**77.62**
		All-Node	**59.64**	**83.00**	83.67	74.03	34.92
SWAT Publication	Precision	End-Node	33.16	41.72	49.51	63.79	**84.13**
		All-Node	**34.67**	**44.35**	**55.23**	**69.15**	80.72
	Recall	End-Node	**100**	**100**	**99.90**	**99.69**	**99.07**
		All-Node	100	99.48	97.09	81.00	42.16
	F1-score	End-Node	49.81	58.88	66.21	**77.80**	**<u>90.99</u>**
		All-Node	**51.48**	**61.35**	**70.41**	74.61	55.39
SWAT Person	Precision	End-Node	13.74	16.93	39.27	83.64	91.09
		All-Node	**13.82**	**17.75**	**44.69**	**85.88**	**92.98**
	Recall	End-Node	**97.85**	97.42	97.42	96.57	96.57
		All-Node	97.42	97.42	97.42	96.57	96.57
	F1-score	End-Node	24.10	28.84	55.98	89.64	93.75
		All-Node	**24.21**	**30.03**	**61.27**	**90.91**	**<u>94.74</u>**

shown in Table 3.1.

In general, only matching end nodes gives better recall while matching all nodes in paths leads to better precision. For RKB publication, RKB person and SWAT publication, the all-node version algorithm has better F1-scores at low thresholds due to its higher precision and comparably good recall; however, for higher thresholds, the end-node version algorithm wins out because of its less affected recall and improved precision. The end-node version algorithm has the best F1-scores for these three datasets. For SWAT person, all-node has the best F1-score because it has better precision than end-node and its recall doesn't get

affected significantly when applying high thresholds. Although end-node is not as good as all-node on SWAT person, only about 1% difference in their best F1-scores was observed; the best F1-scores of end-node are 1.85%, 6.94% and 16.38% higher than those of all-node on RKB publication, RKB person and SWAT publication respectively, still showing its advantage over the other alternative.

3.4.5 Comparing to State-of-the-art Entity Coreference Systems

To further demonstrate the capability of our proposed entity coreference algorithm, we compare E2-P-D to other systems that participated the OAEI2010 (Ontology Alignment Evaluation Initiative 2010) Campaign[6] [50] on the Person-Restaurant (PR) benchmark designed for ontology instance matching. We compare to RiMOM [35], ObjectCoref [34], LN2R [76], CODI [77], and ASMOV/ASMOV_D [78] on the three datasets of PR: Person1, Person2 and Restaurant. Person1 and Person2 are two synthetic datasets where coreferent records are generated by modifying the original records; Restaurant is a real-world dataset, matching instances describing restaurants from Fodors (331 instances) to Zagat (533 instances) with 112 duplicates[7]. Furthermore, we compare to the entity coreference system proposed by Dey et. al. [36] on a synthetic census dataset generated with FEBRL [110]. We compare to these systems by referencing their published results.

In Table 3.2, on Person1 and Person2, our system achieves the best F1-score on both datasets. Although RiMOM, ObjectCoref, LN2R and ASMOV also achieve good results on Person1, their performances drop significantly on Person2. This is due to the difference between how coreferent instances were generated in these two datasets. For Person1, each original instance has at most one coreferent instance with a maximum of 1 modification per

[6]http://oaei.ontologymatching.org/2010/

[7]For full details of these datasets, please refer to the OAEI2010 report [50].

Table 3.2: Comparing to State-of-the-Art Entity Coreference Systems

Dataset	System	*Precision*(%)	*Recall*(%)	*F*1(%)
Person1	E2-P-D	**100**	**100**	**100**
	RiMOM [35]	**100**	**100**	**100**
	ObjectCoref [74]	**100**	99.8	99.9
	LN2R [76]	**100**	**100**	**100**
	CODI [77]	87	96	91
	ASMOV_D [78]	**100**	76.6	87
	ASMOV [78]	**100**	**100**	**100**
Person2	E2-P-D	98.52	**99.75**	**99.13**
	RiMOM [35]	95.2	99	97.1
	ObjectCoref [74]	**100**	90	94.7
	LN2R [76]	99.4	88.25	93
	CODI [77]	83	22	36
	ASMOV_D [78]	98.2	13.5	24
	ASMOV [78]	70.1	23.5	35
Restaurant	E2-P-D	74.58	98.88	**85.02**
	RiMOM [35]	**86**	76.8	81.1
	ObjectCoref [74]	58	**100**	73
	LN2R [76]	75.67	75	75
	CODI [77]	71	72	72
	ASMOV_D [78]	69.6	69.6	69.6
	ASMOV [78]	69.6	69.6	69.6
Census	E2-P-D	**100**	**99.10**	**99.55**
	Dey et. al. [36]	99	98	98.50

coreferent instance and a maximum of 1 modification per attribute of the original instance. Person 2 is created similarly but with a maximum of 3 modifications per attribute, and a maximum of 10 modifications per instance. On the Restaurant dataset, both RiMOM and LN2R achieve better precision than our algorithm, but their recall is much lower than ours. E2-P-D has better precision than ObjectCoref while is only slightly worse on recall. Although E2-P-D is not the best in either precision or recall, it has significantly better F1-score than the other systems. Finally, on the census dataset, our algorithm achieves better performance than that of Dey et. al. on all three metrics.

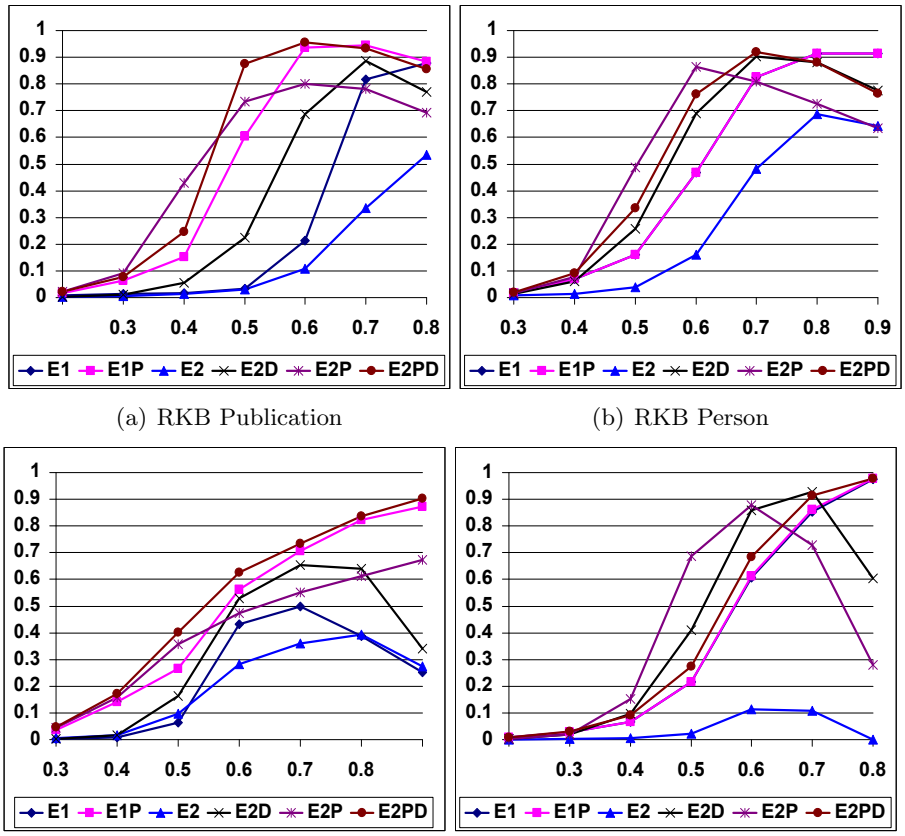
3.4.6 Robustness of Similarity Computations

In our system, we heavily rely on string matching to obtain the similarity between each pair of paths and thus the similarity between two instances. In the results presented in Section 3.4.3, we adopted the JaroWinklerTFIDF string matching algorithm from the secondstring package [108] and we achieved good results. However, string matching algorithms should not be the dominating factor in our system. In this section, we show that our proposed system, E2-P-D, is robust in that it is able to achieve the best performance regardless of the chosen string matching algorithm. Figures 3.5(a) to 3.6(d) show the F1-scores for RKB publication, RKB person, SWAT publication and SWAT person datasets by adopting Edit distance [111] and Jaccard [112] similarity measure respectively.

We can see that our system is not subject to different string matching algorithms. It is true that the achieved F1-scores may vary, the shape of the curves may drift left or right and some other systems are able to achieve equally good results. However, E2-P-D was able to obtain the best F1-scores for all datasets with Edit distance and Jaccard similarity while none of the other comparison systems was always able to achieve this.

3.4.7 System Scalability

In this section, we examine the scalability of our proposed system (E2-P-D). For all experiments in this section, we use Jaccard similarity for string matching. First of all, Figure 3.7(a) shows the runtime by applying our algorithm to 2,000 to 20,000 instances. The y-axis represents runtime in seconds and we use a logarithmic scale with base 10. From the results of RKB person, RKB publication, SWAT person and SWAT publication, we can see that E2-P-D doesn't scale well since essentially it conducts an exhaustive pairwise comparison on all pairs of instances in a given dataset. Considering applying this algorithm to even



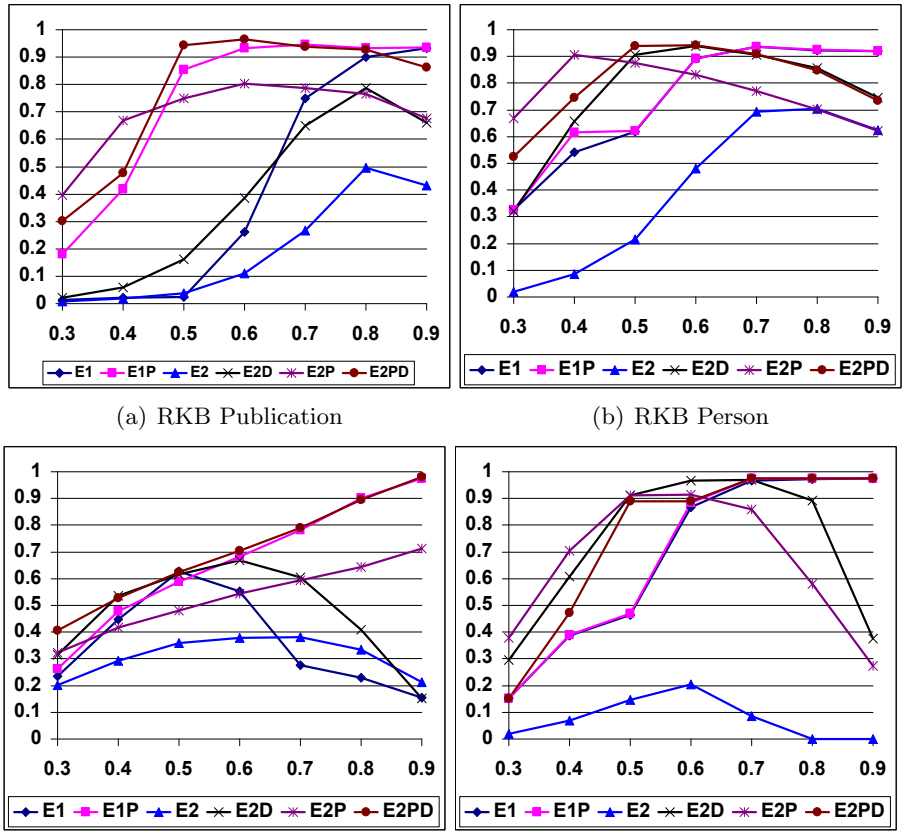
(a) RKB Publication (b) RKB Person

(c) SWAT Publication (d) SWAT Person

Figure 3.5: F1-Scores by adopting the Edit Distance string matching algorithm. In each subfigure, the x-axis is the threshold and the y-axis is the F-score.

larger datasets, techniques that could help to scale the coreference process are desperately needed.

In this scalability test, we parallelized our algorithm on 5 threads. Suppose we have N instances, the number of needed comparisons will be $M = \frac{N*(N-1)}{2}$. We equally divide this M comparisons to the five threads. We put the context information of the N instances into memory and all threads fetch the context information from this in-memory data structure. Figure 3.7(b) shows the speedup factor by distributing the M comparisons to one to eight threads. The speedup factor is computed as $Speedup\_Factor_k = \frac{T_1}{T_k}$, where k is the number of threads and T_k is the runtime by deploying k threads. Except for SWAT Person, the



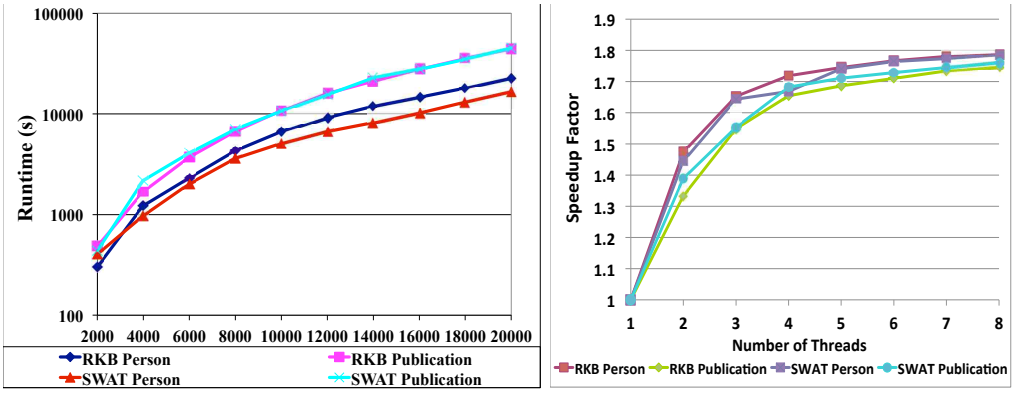
(a) RKB Publication

(b) RKB Person

(c) SWAT Publication

(d) SWAT Person

Figure 3.6: F1-Scores by adopting the Jaccard Distance string matching algorithm. In each sub-figure, the x-axis is the threshold and the y-axis is the F-score.



(a) System Runtime

(b) Speedup Factor with Different Number of Threads

Figure 3.7: System Scalability

other three datasets didn't get significant improvement on this speedup factor by switching from 4 to 5 threads. With more than 5 threads, all datasets continue to achieve higher speedup factors but with diminishing returns.

3.4.8 Discussion

The results show certain advantages of our approach; however, there are a few points to discuss. First, as we described in Section 3.3.5, we are facing the Open World problem. Different subsets of RKB or SWAT may not have complete information for an ontology instance. So, two person instances from ACM RKB and DBLP RKB may be filtered out by applying a high threshold because both of their contexts lack some relevant information. One possible solution to this problem is to merge the contexts of two instances when we have a very high similarity score for them. The intuition behind such merging is to make the context more comprehensive as the algorithm progresses. By employing an iterative entity coreference algorithm, we continue to compare the merged contexts, and therefore could potentially reduce the chance to miss a true match caused by information incompleteness from heterogeneous data sources. However we need to be very careful about doing such merges, since it is easy to add noise to the data if the standard for merging is not appropriately set.

For example, two person instances (a and b) with names *James Smith* and *John Smith* from the same institution co-authored a paper. In this case, the similarity between instances a and b could be high because they share the same last name, work for the same institution and furthermore have the same publication whose title, publication date, venue, etc. information might be available. If we decide to merge the contexts of these two instances to make a combined instance c, the context of c actually contains information of two distinct instances and thus is noisy. In the next iteration, when we compute the similarity between

c and other instances, instances that shouldn't have been merged if individually compared to a and b could be merged due to some matchings provided by the noisy context of c. Iteratively, we could then have contexts that are more and more noisy, which could ultimately affect precision.

Another problem is that, currently, we do not apply penalties for URI mismatches or missing information. This would probably hurt the precision of our system. Syntactically distinct URIs could actually represent distinct real world entities and appropriately penalize instance pairs with distinct URIs in their context could potentially help the system to better differentiate distinct entities. However, applying penalties in such scenarios may cause us to have lower recall. So it is always the problem of keeping a balance between precision and recall. Our choice is not to sacrifice recall while still having a good control on precision by exploiting appropriate weights and context information. One possible way to apply penalties on those situations might be to employ some iterative entity coreference algorithm. At each pass, we record the instance pairs that are clearly not coreferent or clearly coreferent (depending on how the algorithm is designed) and integrate the intermediate results into further iterations until we are only gaining new results under some pre-defined level.

Chapter 4

Context Pruning for Speeding Up Pairwise Entity Coreference

In Chapter 3, we introduced the *EPWNG* algorithm for detecting the coreference relationships between ontology instances and have shown that it is able to outperform several state-of-the-art entity coreference algorithms on small-scale benchmark datasets of different domains. However, the biggest drawback of this algorithm is that it will not scale well to large-scale datasets.

If we assume that multiple heterogeneous sources contain n instances in total, and that the context graphs have branching factor b and depth d, then the time complexity of *EPWNG* is $O(n^2b^{2d})$, making it prohibitively expensive for dealing with large contexts and datasets. On one hand, *EPWNG* compares every pair of instances in a dataset thus making the entity coreference process for large datasets prohibitively expensive. So, one critical question is: ***Can we prune instance pairs that are unlikely to be coreferent to reduce the overall complexity?*** On the other hand, for a pair of instances, *EPWNG* compares all pairs of their comparable paths, therefore making it very time-consuming for

handling large context. So, another interesting question here is: ***Can we only consider the context that could potentially make a significant contribution to the final similarity score between two instances to further speed up the process?***

In the rest of this chapter, I will introduce a context pruning technique to reduce the impact of the size of the context, which enables *EPWNG* to run 30 to 70 times faster.

4.1 Algorithm Design

As described in Section 3.3, we collect a set of paths for an instance. To determine if two instances (x and y from a given RDF graph G) are coreferent, for each path (Path m) of x, the entity coreference algorithm measures m's similarity to all its comparable paths from instance y and picks the maximum to be part of the final similarity measure between the two instances; this process is then repeated for all paths of instance x. Our $EPWNG$ algorithm can be simplified to be Equation 4.1:

$$Sim(x, y) = \frac{\Sigma_{m \in paths}(pw_m * ps_m)}{\Sigma_{m \in paths} pw_m} \tag{4.1}$$

where $paths$ denotes the paths of instance x; m is one of such paths; ps_m and pw_m are the maximum path similarity of path m to its comparable paths from instance y and the corresponding path weight respectively.

In reality, although an instance may have a large number of paths in its context, only those that could potentially make a significant contribution to the final similarity measure should be considered. Based upon this idea, Equation 4.1 is changed to Equation 4.2:

$$Sim(a, b) = \frac{\Sigma_{i \in paths'}(pw_i * ps_i) + \Sigma_{j \in paths''}(pw_j * est_j)}{\Sigma_{i \in paths'} pw_i + \Sigma_{j \in paths''} pw_j} \tag{4.2}$$

where i is one of the paths of instance a that have already been considered by the algorithm; j is one of the paths that have not been covered; est_j and pw_j are the estimated similarity (the potential contribution) of path j to its comparable paths from instance b and its path weight respectively; the combination of $paths'$ and $paths''$ form the entire context of a.

The intuition is that an entity coreference algorithm could safely ignore the rest of the context of an instance when it reaches a boundary. This boundary is a place where the contribution of the remaining context cannot over turn the current decision made based upon the already considered context. In other words, with the estimated path similarity est_j for the remaining paths of instance a, if the similarity between the two instances cannot be greater than a pre-defined threshold, the algorithm should stop at the current path to save computational cost, i.e., it prunes the rest of the context.

Algorithm 4 ComparePruning($samplingOnly, N_a, N_b$), N_a and N_b are the context collected for instances a and b respectively; $samplingOnly$ indicates if the algorithm will use the utility function; returns the similarity between a and b

```
1.  score ← 0, weight ← 0
2.  for all paths m∈N_a do
3.    if Continue(samplingOnly, score, weight, N_a, Pos(m)) then
4.      if ∃path n∈N_b, PtCmp(m, n) then
5.        n ← Comparison(m, N_b);
6.        if n ≠ null then
7.          ps ← Sim(End(m), End(n))
8.          pw ← (W_m + W_n)/2
9.          score ← score + ps * pw
10.         weight ← weight + pw
11.   else
12.     return 0
13. return total_score / total_weight
```

Algorithm 4 shows the pseudo code of the modified algorithm. The key modification is at line 3. The algorithm *Continue* determines if *ComparePruning* should continue to process the next path in the context by estimating the similarity between a and b based on the potential similarity score of the end node of each remaining path in instance a's

context. In the *Continue* algorithm, *Utility* denotes a utility function to determine if it is worth performing such an estimation (line 5-14): 1) if we do not want to use the utility function (i.e., *samplingOnly* is true), we will directly go to line 5 to perform an estimation; 2) if we use the utility function, we calculate the utility of performing an estimation (u). If this utility is less than 0 ($u < 0$), we will return true to go ahead processing the next path; otherwise, we perform an estimation. We shall present the details of the *Continue* algorithm in the rest of this section.

4.2 End Node Similarity Estimation with Random Sampling

Algorithm 5 presents the details of the *Continue* function adopted in Algorithm 4. One problem here is in what ordering should the algorithm consider the paths of an instance. As discussed in Section 3.2, each path has a path weight that is calculated according to the discriminability and the discounting factor of its comprising triples; therefore one approach is to *prioritize* the paths based upon their weight, i.e., paths with higher weight will be considered first. A perfect match on high-weight paths indicates that the algorithm should continue to process the remaining context; while a mismatch on high-weight paths could help the algorithm to stop at appropriate places for non-coreferent instance pairs before wasting more efforts.

Here, *score* and *weight* are the sum of the end node similarity and their corresponding path weight. They represent the similarity (*current*) between two instances based upon the already considered context. When calculating the potential similarity score between the two instances, we only consider the remaining paths whose estimated node similarity ($m'.est$) is no less than *current* (line 7) since paths whose estimated end node similarity is smaller than *current* will only lower the final similarity measure.

Algorithm 5 Continue($samplingOnly, score, weight, N_a, index_m$), $samplingOnly$ indicates if the algorithm will use the utility function; $score$ and $weight$ are the sum of the end node similarity and their corresponding weight of the already considered paths; N_a is the context of instance a; $index_m$ is the index of path m; returns a boolean value

```
1. if samplingOnly is false then
2.    u ← Utility(index_m, |N_a|)
3.    if u < 0 then
4.       return true
5. current ← score/weight
6. for all paths m'∈N_a do
7.    m'.est ← the estimated end node similarity for path m'
8.    if m' has not been considered and m'.est ≥ current then
9.       score ← score + m'.est * m'.weight
10.      weight ← weight + m'.weight
11. if score/weight > θ then
12.   return true
13. else
14.   return false
```

Our entity coreference algorithm (*ComparePruning*) computes coreference relationships by measuring the similarity between end nodes of two paths. So, one key factor to apply our pruning technique is to appropriately estimate the similarity that the last node of a path could potentially have with that of its comparable paths of another instance (*est* in Equation 4.2), i.e., the potential contribution of each path. The higher similarity that a path has, the more contribution it could make to the final score between two instances.

4.2.1 Estimating URI Path Contribution

Since our algorithm only checks if two URIs are identical, Equation 4.3 is used to estimate URI end node similarity for an object value of a set of comparable properties:

$$est(G, P, obj) = \frac{\{t | t =< s, p, obj > \wedge\ t \in G\}}{\{t | t =< s, p, x > \wedge\ t \in G\}}, p \in P \tag{4.3}$$

where G is an RDF graph; P is a set of comparable object properties; obj is a specific object for any property in P; t is a triple and x represents any arbitrary object value of properties

in P. It represents how likely one URI node would meet an identical node in RDF graph G and we calculate it as the estimated similarity for each specific object of property $p \in P$. Similarly, we could compute the estimated similarity for the subject values of all object properties.

4.2.2 Estimating Literal Path Contribution

For literal paths, we could perform such estimation for each specific object value of every predicate. Intuitively, for each object value o of predicate p_i, we calculate its similarity scores to all other object values of the same predicate, draw the distribution of these scores, and choose the most likely score based upon the distribution.

However, considering applying this technique to large datasets where a predicate could be associated with tens of thousands of or even millions of distinct literal values, this naive approach could be extremely expensive. One alternative is that for each object value of predicate p_i, we only compare it to a subset of p_i's object values. This could speed up the processing for each particular object but it will still be expensive when a predicate has a large number of distinct object values. Essentially, this intuitive approach does the estimation for every single distinct value per predicate.

In our approach, to estimate for literal nodes, we randomly select a certain number (ϵ) of literal values of a property, conduct a pairwise comparison among all the selected literals, and finally get the estimated similarity score as shown in Equation 4.4. Here, P

$$est(G,P) = \arg\min_{score} \frac{|\{(o_1,o_2)|o_1,o_2 \in Subset(G,P) \wedge Sim(o_1,o_2) \leq score\}|}{|\{(o_1,o_2)|o_1,o_2 \in Subset(G,P)\}|} > \gamma \quad (4.4)$$

is a set of comparable datatype properties; $Subset(G,P)$ randomly selects some number of literal values of P, such as o_1 and o_2 whose similarity is computed with the function Sim;

γ is a percentage value that controls how many pairwise comparisons should be covered in order to give the estimated node similarity. The intuition here is to find a sufficiently high similarity score as the potential similarity (contribution) between two literal values of P in order to reduce the chance of missing too many true matches. We do not want to set γ too high, since that way we are actually overestimating the potential contribution of paths ending on predicates in P. To summarize, the estimation of literal paths is calculated with respect to each individual property, not for every distinct property value.

One concrete example is given in Figure 4.1 about literal path similarity estimation with Equation 4.4 for the *full_name* predicate in the RKB Person dataset. In this example,

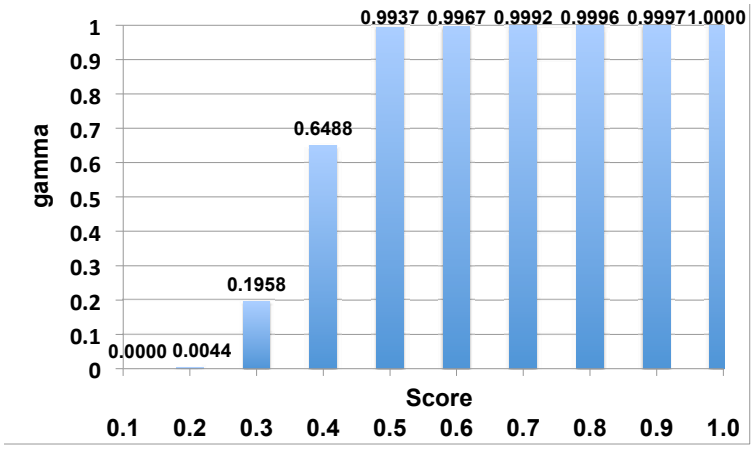


Figure 4.1: Percentage based Path Similarity Estimation

1,000 object values of the *full_name* predicate were randomly selected, and their pairwise similarity scores are discretized to float values from 0.1 to 1 with an interval of 0.1. Figure 4.1 shows the estimated score (x-axis) by considering covering sufficient pairwise comparison scores (y-axis). Looking at the similarity scores from left to right, when reaching 0.5, more than 99% of the pairwise scores have been covered; thus, 0.5 could be used as the estimated contribution for paths that end on the *full_name* predicate.

4.3 Utility Based Decision Making

As described in Algorithm 4, before we actually process each path in the context, we perform an estimation (line 3, *samplingOnly* being true) such that if the potential similarity between two instances would be below a threshold, we just stop considering the rest of the context. However, performing estimations has a computational cost in the entity coreference process. Suppose the algorithm stops after considering k paths, then k estimations were actually performed according to the sampling based technique. However, if the algorithm knew that it would stop after considering k paths, optimally, it should have only performed one estimation at the kth path. To maximally avoid those unnecessary estimations, we ask: ***Can we perform estimations only when needed?***

4.3.1 Utility Function Design

Based upon the discussion above, in order to further reduce the overall complexity of the entity coreference process, we define a utility function as shown in Figure 4.2. Suppose we

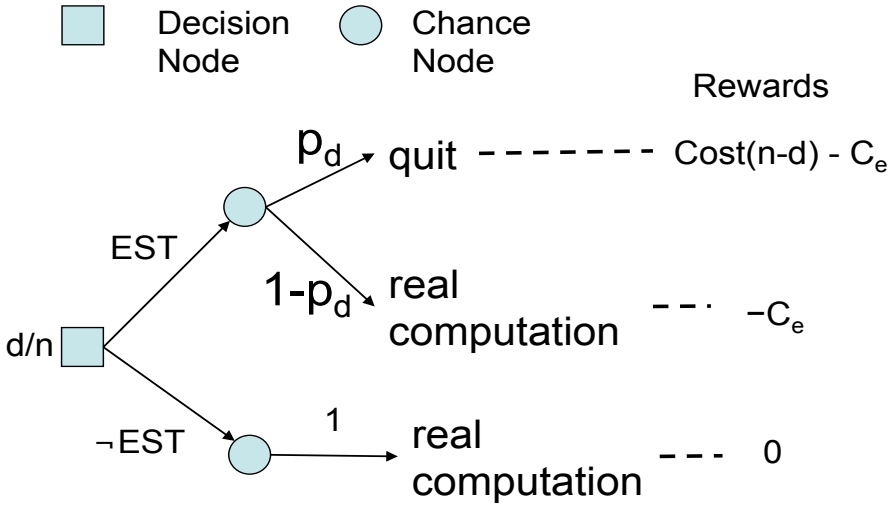


Figure 4.2: Utility Function

reach a decision node d/n (there are n paths in total and the algorithm is now at the dth

path), we would then need to decide whether we want to perform an estimation (EST vs. $\neg EST$). The general decision making process is described as follows:

- If we choose not to do estimation ($\neg EST$), then the only choice is to perform a real computation. For a path m of instance a, the algorithm will find all paths comparable to m from the context of instance b and compute the similarity by following line 4 to 10 in Algorithm 4. We will not have any rewards by going this route;
- If we perform an estimation at node d/n (EST), then there could be two different outcomes as follows:
  - One possibility is that the algorithm will quit (with probability p_d) because it estimates that this pair of instances are not similar to a pre-defined level (their estimated similarity score is lower than θ in Algorithm 5). In this case, we have rewards $Cost(n-d)$, where n is the total number of paths of instance a, d is the current path number, $Cost(n-d)$ is the cost of performing real computations for the rest $n-d$ paths, i.e., $Cost(n-d) = (n-d) * C_r$ with C_r being the cost of doing a real computation for a path. Here, we also spent some time (C_e) for an estimation;
  - If the algorithm continues based upon the estimation results, then the rewards will be $-C_e$ because there is no gain but an estimation has been performed.

Summarizing all these possibilities, the utilities for actions EST and $\neg EST$ are formally defined in Equations 4.5 and 4.6 respectively:

$$EU(EST) = p_d * (Cost(n-d) - C_e) + (1-p_d) * (-C_e) \tag{4.5}$$

$$EU(\neg EST) = 0 \tag{4.6}$$

where p_d is the probability that the algorithm stops at the dth path. We perform an estimation when the marginal utility given in Equation 4.7 is a positive value (line 2 of Algorithm 5) otherwise a real computation is executed.

$$\begin{aligned} Utility = EU(EST) - EU(\neg EST) = \\ p_d * (n-d) * C_r - p_d * C_e - C_e + p_d * C_e = \\ p_d * (n-d) * C_r - C_e \end{aligned} \tag{4.7}$$

4.3.2 Parameter Value Estimation

To estimate the parameters for each category of instances, we randomly select a small number of instances (α) for each category and run the sampling based algorithm on them. Then, we compute the average time for performing an estimation (C_e) and a real computation (C_r) respectively. We adopt Equation 4.8 to estimate the probability that the algorithm stops at the dth path (p_d):

$$p_d = \frac{|The\ algorithm\ stopped\ at\ the\ dth\ path|}{|The\ algorithm\ passed\ d\ paths|} \tag{4.8}$$

One potential issue is that the estimated p_d may not be accurate when d is large. Typically, we would expect the algorithm to stop far before processing all the paths; therefore, for large d values, there may not be enough samples to generate convincing probabilities.

4.4 Evaluation

In this section, we evaluate our sampling and utility function based pruning methods. We first introduce our evaluation datasets and metrics and then present the evaluation results.

4.4.1 Evaluation Datasets, Metrics and Methodology

Datasets and Metrics. We evaluate our pruning techniques on two RDF datasets: RKB [17] and SWAT[1], that were adopted in the previous chapter (Section 3.4.1). In order to examine how the system will perform by applying our pruning techniques, we compare on larger testing sets. We randomly selected 100K instances for each of the three instance categories: RKB Person, RKB Publication and SWAT Person. We measure the precision, recall, F1-score and runtime for our proposed algorithm on the three testing sets.

Evaluation Methodology and Comparison Systems. We compare the performance of 3 systems: *EPWNG* [16], the sampling based algorithm (*Sampling*) and the system that combines sampling and the utility function (*Utility*). We report a system's best F1-Score from threshold 0.3-0.9. We split each 100K dataset into 10 non-overlapping and equal-sized subsets, run all algorithms on the same input and report the average. We also test the statistical significance on the results of the 10 subsets from two systems via a two-tailed t-test. On average, there are 6,096, 4,743 and 684 coreferent pairs for each subset of RKB Person, RKB Publication and SWAT Person respectively.

[1]http://swat.cse.lehigh.edu/resources/data/

4.4.2 Parameter Settings

We have the following parameters in our system: γ (Equation 4.4) is the percentage of pairwise similarities to be covered to estimate literal node similarity; θ (Algorithm 5) determines if an instance pair is still possible to be coreferent; ϵ is the number of literal values selected for literal node similarity estimation; and α is the number of instances to be used for estimating the parameters in our utility function. We set γ, θ, ϵ and α to be 0.95, 0.2, 1,000 and 200 respectively and use the same values for all experiments. It took 110 and 78 seconds to estimate literal end node similarity for the RKB dataset and the SWAT dataset respectively.

4.4.3 Entity Coreference Results with Sampling and Utility Function based Pruning Techniques

Table 4.1 shows the evaluation results by applying the three different entity coreference algorithms on the ten subsets of the three 100K subsets. We also compare to a baseline where we simply choose the top $\beta\%$ most disambiguating context.

With our proposed pruning techniques, both *Sampling* and *Utility* run much faster than $EPWNG$. Particularly, with the utility function, the system *Utility* was able to achieve even more runtime savings than *Sampling*. On the other hand, while successfully scaling the $EPWNG$ algorithm, both *Sampling* and *Utility* still maintain comparable F1-scores to that of $EPWNG$ on all datasets. Interestingly, they both achieve even higher F1-scores than $EPWNG$ on RKB Person. The improvements here come from higher precision since the pruning techniques can help to remove some paths where non-coreferent instances happen to have similar values (e.g., two distinct person instances could have identical date for their publications). Although the F1-score of *Sampling* and *Utility* is slightly lower than that of

Table 4.1: Entity Coreference Results. *Precision* and *Recall* represent precision and recall respectively; F1 is the F1-score for *Precision* and *Recall*; Baseline(β%) means only using top β% weighted paths

Dataset	System	*Precision*	*Recall*	*F*1	*Total Time* (s)
RKB Person	EPWNG	93.04	**90.93**	91.96	6,296.91
	Baseline (5%)	**94.84**	88.82	91.72	**182.96**
	Baseline (10%)	94.18	89.4	91.72	304.66
	Baseline (20%)	93.82	89.18	91.43	1143.13
	Sampling	94.22	90.41	92.27	246.26
	Utility	94.22	90.45	**92.29**	215.50
RKB Publication	EPWNG	99.78	99.37	99.58	63200.71
	Baseline (5%)	99.78	99.28	99.53	**760.90**
	Baseline (10%)	99.8	**99.44**	**99.62**	1577.50
	Sampling	99.45	99.00	99.22	921.74
	Utility	**99.82**	99.30	99.56	828.56
SWAT Person	EPWNG	**99.45**	90.93	**94.99**	16968.00
	Baseline (5%)	99.37	90.93	94.95	286.64
	Baseline (10%)	99.35	90.94	94.96	892.06
	Baseline (20%)	99.29	**90.96**	94.93	3,911.65
	Sampling	**99.45**	90.93	**94.99**	307.42
	Utility	**99.45**	90.93	**94.99**	**275.17**

EPWNG on RKB Publication, a statistical test on the F1-scores shows that the difference between *EPWNG* and *Sampling*/*Utility* on RKB Publication is not significant with P values of 0.3240 and 0.3945 respectively.

Compared to the baseline, *Utility* achieves better F1-scores on both person datasets, though it is not as good as the baseline on RKB Publication. On RKB Person, when β=5%, the baseline was able to finish faster than *Utility* but with lower coreference F1-score. When adopting higher β values, the baseline was overwhelmed by the low-weighted context information and thus was not able to continue to achieve even higher F1-score; in the meantime, as it considers more context, it needed much long runtime to finish than *Utility*. On both RKB Publication and SWAT Person, the baseline (β=5%) achieves a comparable or even slightly better runtime than *Utility* with minor difference in F1-score. One possible reason could be that these two datasets are not as ambiguous as RKB person; thus simply

cutting off those redundant paths by setting appropriate thresholds could greatly speed up the process while still achieving good F1-scores.

Although we see significant speedup by applying our proposed pruning techniques, problems still exist. The most critical issue is still about scalability. Extrapolating from the runtime on 10K instances, it will take around 36 hours to finish processing one million instances. Considering the fact that we already have more than 100 million instances in the BTC dataset, the current system is not sufficiently efficient. To further improve the scalability of our entity coreference system, we will propose two other techniques that can help us to achieve toward the goal of handing really large-scale datasets.

Chapter 5

Accuracy vs. Speed: Scalable Entity Coreference with On-the-Fly Candidate Selection

The biggest problem of the previous algorithms is that they perform an exhaustive pairwise comparison for every pair of instances, although some pruning techniques were applied to the process. However, in a large-scale dataset with millions of instances, it would be nearly impossible to perform pairwise comparison for every pair of instances in the dataset. Computing the similarity of two un-coreferent instances is a waste of computing effort without contributing anything to the final coreference results. To further improve the scalability of our entity coreference system, we propose an on-the-fly candidate selection algorithm to reduce the impact of the number of instances in a dataset. Candidate selection refers to the process of selecting pairs of instances that could be coreferent, and then more expensive entity coreference algorithms (e.g., *EPWNG*) will only be applied to the selected pairs to save the overall computational costs. This candidate selection technique is also

frequently referred to as *Blocking* in the database research field [43, 40, 41].

In this chapter, we describe in detail a pruning-based algorithm for reducing the complexity of entity coreference within a dataset itself. Here, a "dataset" may contain comparable instances from multiple heterogeneous sources. For example, different datasets may use different terms to define the "Researcher" class and ontology instances of all such classes are comparable. First, we design an on-the-fly candidate selection technique to reduce the number of instance pairs to be computed. During the entity coreference process, each instance is compared against other instances; and we hypothesize that two coreferent instances should have similar matching histories, i.e., they should be similar to a sufficiently common set of other instances. We further propose a sigmoid function based thresholding method to automatically adjust the threshold on such history similarity in order to gain a good balance between runtime and F1-score. To speed up the computation for a single pair of instances, we evaluate the potential contribution of their context and only consider the context that is likely to make a significant contribution to their final similarity measure. By comparing to 9 state-of-the-art systems on three Semantic Web datasets, we show that our algorithm runs 16 to 22 times faster with comparably good F1-scores. A scalability test shows that the runtime difference becomes even more substantial as we increase the size of the datasets.

5.1 Overview of the On-the-Fly Candidate Selection Algorithm

Algorithm 6 presents the pseudo-code of our proposed pruning-based entity coreference algorithm *P-EPWNG*. The proposed candidate selection algorithm consists of the following

primary components and I will detail each individual component in the rest of this section:

- *Share_A_Token_Cosine*: This step is used to filter instance pairs that do not even share a single token and the cosine similarity of their most disambiguating context is below a pre-defined threshold;
- Computing the similarity between the k% most disambiguating context of a pair of instances: This step determines if one instance should be added to the history of another;
- *EPWNG*: We finally adopt the previously discussed *EPWNG* algorithm to determine the final similarity score between two instances with their full context.

At line 1 of Algorithm 6, we adopt a simple yet effective filtering method, i.e., we check if two instances share a single token in the literals of their top k% most disambiguating context, and if the cosine similarity of such instances' top k% context is above a threshold δ. First, two instances cannot be coreferent if they do not share at least one common word in their context; also, the cosine similarity is used to better handle contexts with many tokens. For example, for publication instances, their titles are generally very disambiguating and thus included in the top k% context; however, only sharing a single token does not necessarily indicate that they could be coreferent in many cases.

From line 3 to 10, we propose an on-the-fly candidate selection technique based upon instances' matching histories to effectively prune out instance pairs that are not likely to be coreferent. Consider that during the entity coreference process, an instance is compared to many other instances; the results of these prior comparisons could be useful in determining whether two instances might be coreferent. At any point in time, each instance should have a set of other instances that it is somewhat similar to. We define a function $H(a)$ to denote

Algorithm 6 P-EPWNG($N_a, N_b, H(a), H(b)$), N_a and N_b are the context for instances a and b; $H(a)$ and $H(b)$ are their histories; returns their similarity

```
1. if Share_A_Token(N_a, N_b, k) = false then
2.    return 0;
3. if HSim(H(a), H(b)) ≤ Thresholding(a) then
4.    return 0;
5. else
6.    sim_k ← EPWNG(kdisc(N_a, k), kdisc(N_b, k))
7.    if sim_k ≤ θ then
8.       return 0;
9.    else
10.      H(b) ← H(b) ∪ {a}
11. score ← 0, weight ← 0, s_curr ← 0
12. context_signif ← γ
13. for all paths m∈N_a do
14.    n ← Compare(m, N_b);
15.    if n ≠ null then
16.       ps ← Sim(End(m), End(n))
17.       pw ← (W_m + W_n)/2
18.       s_old ← s_curr
19.       score ← score + ps * pw
20.       weight ← weight + pw
21.       s_curr ← score/weight
22.       context_signif ← context_signif − Eval(s_old, s_curr)
23.       if context_signif = 0 then
24.          return s_curr
25. return score/weight
```

the set of similar instances of instance a, i.e., the matching history. We hypothesize that two coreferent instances should share a sufficient amount of common instances in their histories. Therefore, the general idea behind our on-the-fly candidate selection technique is that before we actually compute the similarity for instances a and b, we adopt a lightweight method to examine if their histories are similar enough, i.e., if $HSim(H(a), H(b))$ is above some threshold (line 3). Instead of utilizing a fixed threshold on $HSim$, we propose a sigmoid function based thresholding method that gradually increases the threshold at runtime until an upper bound is reached, since there is very little history to compare at the beginning. $HSim$ and the thresholding method are further presented in Section 5.2.

In the next step, from line 6 to 10, after an instance pair passes the history check, we should then apply a more expensive method to measure their similarity, which can also be used to determine if we should add one instance to the history of another. In our algorithm, at line 6, we make this decision by computing two instances' similarity by using the *EPWNG* algorithm only with their top k% most disambiguating context (as returned by $kdisc(N_a, k)$). Passing the history check does not necessarily mean the two instances are coreferent and thus it might not make sense to use full context.

Finally, we propose a context pruning technique inspired by the idea of quiescence search in game playing. The basic idea is that more time should be spent refining heuristic evaluations when there appear to be significant fluctuations in the value. In this work, we define an evaluation function to estimate if the remaining context would still make a significant contribution to the final similarity score of two instances. At line 12, we initialize $context\_signif$ (context significance level) with a positive integer γ that can be interpreted as the maximum times that we allow a new path to provide only a small contribution to the final similarity score. Then, at line 22, an evaluation function *Eval* is used to check the difference between s_{curr} and s_{old}, i.e., if considering one more path still makes significant changes to the similarity score of two instances. We prune the rest of the context when $context\_signif$ reduces to 0. *Eval* is further discussed in Section 5.3.

5.2 Measuring Instance Matching History Similarity with Sigmoid Function based Thresholding Method

5.2.1 Measuring the Similarity between Instances' Matching Histories

As shown in Algorithm 6, we compute the similarity ($HSim$, line 3) between the matching histories of two instances. In our approach, we adopt a modified version of the well-utilized Jaccard similarity measure [112] as defined in Equation 5.1:

$$HSim(H(a),\, H(b)) = \begin{cases} +\infty, & H(a)=\emptyset \text{ or } H(b)=\emptyset \quad (5.1a) \\ \dfrac{|H(a) \cap H(b)|}{|H(a) \cup H(b)|}, & \text{otherwise} \quad (5.1b) \end{cases}$$

where $H(a)$ represents the matching history of instance a, i.e., a set of other instances that a is similar to. Here, $HSim$ is $+\infty$ when either instance has an empty history, though the traditional Jaccard similarity measure will give a 0 in this case. The intuition is that the history we use for pruning is partial history in the sense that it does not contain a complete set of other instances that the given instance is similar to. Imagine we have a list of instances as shown below:

$$I = < i_1,\ i_2,\ i_3,\ ...,\ i_m,\ i_n,\ ...\ ,\ i_{last} >$$

To detect coreference relationships, we compare each instance with every other instance in the order they are placed, i.e., compare i_1 *to* $i_2,\ i_3,\ ...,\ i_{last}$ and then similarly compare i_2 to all other instances after it. Therefore, when we attempt to compare i_m and i_n, each of them has only been compared with instances that are placed to the left of them and we still do not know their similarities to instances in $\{i_k\}$ (k>n). Due to such incomplete knowledge, an instance pair should not be pruned when seeing an empty history.

5.2.2 Sorting Instances as Pre-processing

At Line 3 of Algorithm 6, we prune a pair of instances if their history similarity is below a threshold. In our algorithm, this threshold is automatically adjusted at runtime. One approach for performing such adjustment is based upon how many groundtruth coreferent pairs have already been covered by the processed instances at any given time. When more coreferent pairs have been covered, we should adopt a higher threshold in order to prune more un-coreferent pairs. This will lead to runtime savings, without the risk of missing too many true matches. However, for this approach to work, we need a sufficiently sized sample of groundtruth data, but obtaining such a sample for large-scale data is impractical.

To avoid labeling, we compute a ***Match Heuristic*** (MH) for each instance before executing Algorithm 3. MH is the number of potential matches of each instance in the dataset. We sort the instances by their MH in descending order, and we call Algorithm 3 on all pairs of an instance with a subsequent instance in descending MH order. To compute MH, we treat the context of each instance as a *bag of words* that we call *doc*. For an instance, *doc* is extracted from all literal paths of length one, i.e., literal values from immediate triples. We then compute a cosine similarity between the *doc* of each instance and that of all other instances with Lucene[1], a well-adopted Information Retrieval tool. We first index all the *doc*s as documents with Lucene. Then each document is treated as a query q and issued to Lucene. We count a returned document doc' as a potential match if $cosine(q, doc')$ calculated by Lucene is above a threshold α.

[1]http://lucene.apache.org/core/3_6_0/api/core/org/apache/lucene/search/Similarity.html

5.2.3 A Sigmoid Function based Thresholding Approach

There could be different ways to adjust the threshold on the similarity of two instances' matching histories. The simplest way is to use a fixed threshold for the entire coreference process; however, this method may cause the system to miss a certain amount of coreferent pairs at the starting stage since we put instances with the most potential matches at the beginning of the instance list. Therefore, a thresholding approach that starts with a low value but then gradually increases the threshold could potentially be a good choice.

After sorting, each instance is associated with a match heuristic. Intuitively, the match heuristic of an instance can be considered as its weighting factor. When processing instance i_m, $\Sigma_{k=1}^{m-1} MH(i_k)$ gives the total weight of the already processed instances (i.e., their total match heuristics), indicating how important they are and thus can be helpful for thresholding.

Based upon the discussion above, we propose a sigmoid function based thresholding method that utilizes such match heuristics. The traditional sigmoid function is defined by Equation 5.2 and illustrated in Figure 5.1. The traditional sigmoid function provides a continuous output with limit 1 as x approaches infinity, and limit 0 as x approaches negative infinity.

$$F(x) = \frac{1}{1 + e^{-x}} \tag{5.2}$$

However, the traditional sigmoid function cannot be directly applied to our situation due to the following two aspects. First of all, the traditional function gives a value of 0.5 when x=0; while we require the computed threshold should be 0 when input is 0. When x is small, the entity coreference process is still in its early stage and thus we have very limited knowledge about the matching histories of the instances. Also, since we sort instances based

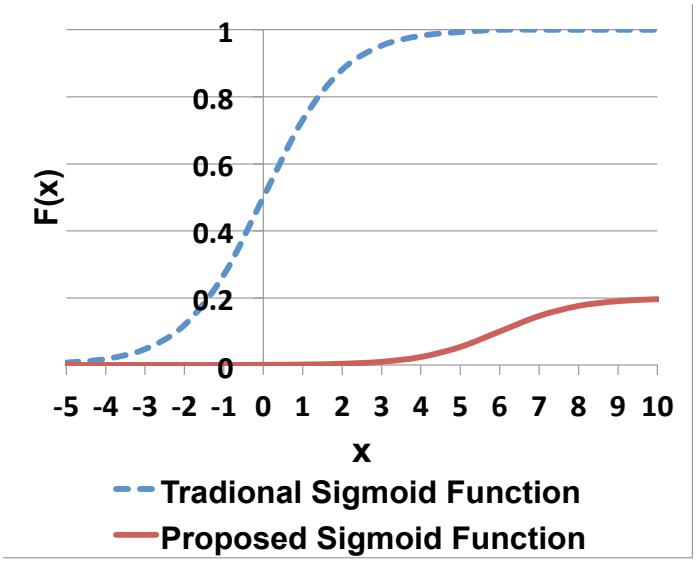


Figure 5.1: The Traditional and Proposed Sigmoid Curves

upon the number of their potential matches in the dataset in descending order, we want to adopt a relatively low threshold at the early stage in the entire entity coreference process to reduce the risk of missing too many true matches. Furthermore, the traditional function has an upper bound of 1. However, in our situation, we may not want to set such a high threshold for filtering and it would be useful if we could adjust the upper bound by setting some parameter.

Based upon the discussion above, we propose a two-phased thresholding function as shown in Equation 5.3:

$$Thresholding(i_m) = \begin{cases} 0, & x \leq \sigma \quad (5.3a) \\ K * F(x(i_m) - 6), & x > \sigma \quad (5.3b) \end{cases}$$

where σ determines when to start introducing a threshold; K represents the upper bound of the computed threshold. Input x is the ratio between match heuristics of already processed

instances and that of all instances as computed in Eq. 5.4:

$$x(i_m) = \frac{\Sigma_{k=1}^{m-1} MH(i_k)}{\Sigma_{k=1}^{|I|} MH(i_k)} * 10 \tag{5.4}$$

where $MH(i)$ is the match heuristic for instance i; the factor 10 enables x to have the range [0, 10] to ensure the output (the computed threshold by Eq. 5.3) has the ability to transition through most of the range. We shift the original sigmoid curve to the right by replacing x in Equation 5.2 with x-6, thus the computed threshold gets very close to 0 when x=0.

One alternative to Equation 5.3 is *Bump*, i.e., setting the history threshold to 0 at the beginning and bumping it to the upper bound when $x>\sigma$. We shall compare these different thresholding approaches in our evaluation.

5.3 Evaluation Function based Context Pruning

Another important question discussed in Chapter 2 is how to reduce the complexity of the comparison for a single pair of instances. Although an instance may have a large number of paths in its context, we should only consider those that can actually make significant contribution to the final similarity score of two instances. Therefore, we define an evaluation function in Equation 5.5 to judge if we should continue with the remaining paths given the similarity between two instances from their already considered paths:

$$Eval(s_{old},\ s_{curr}) = \begin{cases} 1, & s_{curr} - s_{old} \leq \beta \quad (5.5a) \\ 0, & otherwise \quad (5.5b) \end{cases}$$

where s_{old} is the similarity between two instances by considering the most highly weighted n-1 paths, and s_{curr} is the similarity by considering the top n paths.

Our evaluation function provides a hint of how important the remaining context is. Linking back to Algorithm 6, at line 12, $context\_signif$ (context significance level) is initialized with γ, the maximum times we can tolerate when considering one more path does not significantly increase the similarity of two instances, i.e., s_{curr}-$s_{old} \leq \beta$. When this happens, $Eval$ will return 1 and consequently at line 22, we reduce our tolerance level of such situations by 1. When the context significance level reaches 0, we suspect all remaining context is insignificant, and we will then decide to ignore the rest of the context to save the overall computational cost. We will show later that adopting our evaluation function can save half of the overall runtime.

5.4 Evaluation

In this section, I present the evaluation results of the discussed on-the-fly candidate selection technique. We conduct all experiments on a RedHat machine with a 12-core Intel 2.8GHz processor and 60GB memory.

5.4.1 Evaluation Datasets, Metrics and Methodology

The RKB [17] and SWAT[2] datasets are still adopted. Similarly, we continue to empoly the standard metrics, including *Precision*: the number of correctly detected pairs divided by the total number of detected pairs given some threshold; *Recall*: the number of correctly detected pairs divided by the number of coreferent pairs according to the groundtruth; and *F1-score* calculated as $2 * \frac{Precision * Recall}{Precision + Recall}$. Furthermore, we compare the runtime of different

[2]http://swat.cse.lehigh.edu/resources/data/

systems.

Evaluation Methodology. We compare *P-EPWNG* to state-of-the-art candidate selection systems: *AllPairs* [97], *PPJoin+* [100], *EdJoin* [99], *FastJoin* [96], *IndexChunk* [95], *BiTrieJoin* [94] and *PartEnum* [93]. Although *P-EPWNG* performs both candidate selection and entity coreference, traditional candidate selection systems only do the first step. Thus, to compare to state-of-the-art systems, we first run those systems on the same input to select candidate instance pairs; and then we apply the *EPWNG* algorithm (the algorithm that does not have our proposed pruning techniques) to the selected pairs. We directly run our proposed algorithm *P-EPWNG* on the same input to get the final entity coreference results.

We also compare to *EPWNG* [16] that performs a brute-force pairwise comparison on all instance pairs without any pruning and to *U-EPWNG* [19] that prunes insignificant context based upon sampling techniques and a utility function without doing candidate selection. A baseline is also tested against where we only compare name (person) and title (publication) for every pair of instances.

We randomly select 100K instances for each instance category. We split each 100K instances into 10 non-overlapping and equal-sized subsets, apply all algorithms on each subset and report their average. For coreference results, we report a system's best F1-score from threshold 0.1-0.9.

5.4.2 Parameter Settings

For all experiments, we use the same parameter settings as shown in Table 5.1. Adopting higher k improves F1-score a little but slows down the process; using lower γ causes the

Table 5.1: Parameter Settings.

Parameter	Value	Description
k (Alg. 6)	5%	the percentage of context used for filtering
θ (Alg. 6)	0.3	whether to add one instance to the history of another
γ (Alg. 6)	5	the maximum tolerance level for context pruning
α (Sec. 5.2)	5	sorting: whether an instance is a potential match
K (Eq. 5.3)	0.2	the upper bound threshold for history-based pruning
σ (Eq. 5.3)	30%	determining when to apply history based pruning
β (Eq. 5.5)	0.1	the expected contribution of computing one more path

system to have lower F1-score by pruning significant context; using a higher σ, i.e., introducing history-based filtering later, typically improves the F1-score. We will present a more comprehensive evaluation for different parameter settings later in this section.

5.4.3 Evaluating Against Comparison Systems

We compare our proposed on-the-fly pruning techniques to our previous $EPWNG$ algorithm and the baseline system and show the comparison results in Table 5.2.

Baseline. The baseline is not as good as *P-EPWNG* on F1-score and runtime. It has a much lower recall than others on RKB Person. On RKB publication, the baseline has worse precision than *P-EPWNG* since candidate selection helps to filter out some potential false positives. Although we only observed minor difference on F1-score for SWAT Person, the baseline needed significantly more time. For both RKB datasets, both *P-EPWNG* and the baseline have worse recall than *EPWNG*, showing the need to explore context beyond just name and title and thus the need for appropriate context pruning techniques to balance runtime and F1-score. Note that the baseline requires human input on what information to compare; also, in the absence of discriminative labels (such as name and title), it may not be able to achieve satisfying results.

EPWNG and U-EPWNG. Overall, compared to our previous entity coreference

Table 5.2: Evaluating Against Comparison Systems and the Baseline System. *Precision*, *Recall* and *F*1-*score* are the relevant measures for the actual entity coreference phase; $Time$: the runtime for the entire process, including both candidate selection and entity coreference.

Dataset	System	$Coreference$			$Time(s)$
		Precision (%)	*Recall* (%)	*F1* (%)	
RKB Person	P-EPWNG (Eq. 5.3)	95.02	89.52	92.18	**7.66**
	Baseline	**95.47**	83.66	89.15	36.69
	EPWNG [16]	93.04	**90.93**	91.96	6,296.91
	U-EPWNG [19]	94.22	90.45	**92.29**	215.50
	S-EPWNG [19]	94.22	90.41	92.27	246.26
SWAT Person	P-EPWNG (Eq. 5.3)	**99.49**	90.88	**94.99**	**8.61**
	Baseline	99.37	**90.93**	94.96	31.7
	EPWNG [16]	99.45	**90.93**	**94.99**	16968.00
	U-EPWNG [19]	99.45	**90.93**	**94.99**	275.17
	S-EPWNG [19]	99.45	**90.93**	**94.99**	307.42
RKB Pub	P-EPWNG (Eq. 5.3)	99.66	98.10	98.87	**23.06**
	Baseline	98.99	97.28	98.13	72.1
	EPWNG [16]	99.78	**99.37**	**99.58**	63200.71
	U-EPWNG [19]	**99.82**	99.30	99.56	828.56
	S-EPWNG [19]	99.45	99.00	99.22	921.74

algorithms *EPWNG* and *U-EPWNG*, *P-EPWNG* achieves substantial runtime savings with slightly worse recall and F1-score. The coreference process was sped up by 2-3 orders of magnitude compared to *EPWNG* and by a factor of 28-36 compared to *U-EPWNG*. There is no surprise that *EPWNG* always achieves the highest recall since no pruning was adopted. Also, *P-EPWNG* achieves a little better precision on both person datasets because pruning dissimilar instance pairs can help to reduce the chance of having false positives.

5.4.4 Evaluating Against State-of-the-Art Candidate Selection Systems

Table 5.3 shows the evaluating results when compared to state-of-the-art candidate selection systems. Here, we compare the results of both the candidate selection phase and the actual coreference phase.

Table 5.3: Candidate Selection Results on RDF Datasets. $|Pairs|$: candidate set size; CST: time for candidate selection; RR: Reduction Ratio; PC: Pairwise Completeness; F_{cs}: the F1-score for RR and PC; P, R and $F1$ are the relevant measures for the actual entity coreference phase; Total: the runtime for the entire entity coreference process

Dataset	System	*Candidate Selection*					*Coreference*(%)			Time (s)
		$\|Pairs\|$	CST	$PC(\%)$	$RR(\%)$	$F_{cs}(\%)$	P	R	F1	$Total$
RKB Person	PEPWNG [16]	68,427	4.46	94.94	**99.86**	97.34	95.02	89.52	92.18	**7.66**
	Ed-Join [99]	207,643	1.31	99.60	99.58	**99.59**	95.06	90.74	**92.84**	63.31
	AllPairs [97]	454,972	**0.93**	**99.64**	99.09	99.36	94.26	**90.85**	92.52	83.76
	PPJoin+ [100]	454,972	1.02	**99.64**	99.09	99.36	94.26	**90.85**	92.52	82.96
	FastJoin [96]	443,846	3.92	**99.64**	99.11	99.38	94.27	**90.85**	92.52	81.75
	IndexChunk [95]	953,300	2.27	99.37	98.09	98.73	95.03	90.73	92.82	149.70
	BiTrieJoin [94]	969,637	41.92	91.46	98.06	94.63	**95.65**	84.61	89.77	87.66
RKB Publication	PEPWNG [16]	107,471	13.18	98.27	**99.79**	99.02	**99.66**	98.10	98.87	**23.06**
	Ed-Join [99]	1,298,525	131.85	98.54	97.40	97.97	99.45	98.36	98.90	1330.20
	AllPairs [97]	581,005	1.41	99.21	98.84	99.02	99.47	99.06	**99.27**	340.14
	PPJoin+ [100]	581,005	**1.39**	99.21	98.84	99.02	99.47	99.06	**99.27**	342.21
	FastJoin [96]	583,839	75.78	**99.22**	98.83	**99.03**	99.47	**99.07**	**99.27**	430.61
	IndexChunk [95]	1,594,764	156.26	98.20	96.81	97.50	99.44	98.05	98.74	1242.85
SWAT Person	PEPWNG [16]	43,333	4.50	96.79	**99.91**	98.32	99.49	90.88	**94.99**	**8.61**
	Ed-Join [99]	226,330	1.56	99.63	99.55	**99.59**	99.48	90.81	94.94	102.77
	AllPairs [97]	381,128	**0.79**	**99.80**	99.24	99.52	99.45	**90.93**	**94.99**	108.34
	PPJoin+ [100]	381,128	0.86	**99.80**	99.24	99.52	99.45	**90.93**	**94.99**	106.72
	FastJoin [96]	360,481	3.20	**99.80**	99.28	99.54	99.45	**90.93**	**94.99**	103.72
	IndexChunk [95]	562,114	1.67	99.25	98.88	99.06	99.48	90.84	94.96	210.96
	BiTrieJoin [94]	472,875	22.13	96.67	99.05	97.84	**99.61**	90.29	94.72	124.02

Generally speaking, when compared to state-of-the-art candidate selection systems, *P-EPWNG* selects the fewest candidate pairs and thus is able to run the fastest for the entire process, including both candidate selection and coreference. However, since *P-EPWNG* is the most aggressive in pruning instance pairs, its recall and F1-score for the final coreference results are generally slightly worse than that of most of the state-of-the-art systems.

On RKB Person, *P-EPWNG* selects 3 (*EdJoin*) to 13 (*IndexChunk*, *BiTrieJoin* and *PartEnum*) times fewer pairs than other systems, which leads to a speedup factor of 7 (*EdJoin*) to 16 (*IndexChunk*) on runtime but also results in 0.83% lower F1-score than the best (achieved by *EdJoin*). Although *P-EPWNG* needs more time for candidate selection, the fact that it selects a lot fewer candidates enables it to achieve substantial runtime savings for the overall process.

We observe similar results for SWAT Person. *P-EPWNG* selects about 5 (*EdJoin*) to 14 (*IndexChunk*) times fewer candidates than other systems; thus the entire process runs 10 (*EdJoin*) to 20 (*IndexChunk*) times faster. Although *P-EPWNG* has a little lower recall, compared to *AllPairs*, *PPJoin+* and *FastJoin*, it achieved identical F1-score due to a little better precision.

On RKB Publication, in addition to runtime savings, *P-EPWNG* also achieves the highest precision among all candidate selection systems. *P-EPWNG* prunes instance pairs most aggressively, which helps to reduce the chance of having too many false positives. Note that on this dataset, *P-EPWNG* spends much more time selecting candidates than needed for the two person datasets because our *Share_A_Token* filtering is not very effective here. *Share_A_Token* prunes an instance pair if they do not even share a single token in their most disambiguating context. For person and publication instances, names and titles are generally included in the partial context used by *Share_A_Token*; and it is relatively more

common to share a token in titles than in names. So, more pairs can pass this check and are then processed by other more expensive steps.

Our proposed *P-EPWNG* algorithm enables the entire coreference process to run 18 to 24 times faster than state-of-the-art candidate selection systems with a small sacrificing in its coreference F1-scores. However, the small sacrificing on the final coreference F1-scores may not accurately reflect the candidate selection capability of an algorithm. It could be the case that the candidate selection algorithm does miss a significant number of coreferent instance pairs according to the groundtruth while these missing pairs just cannot be detected by the actual coreference algorithm even they were included. To examine the capability of our proposed candidate selection algorithm in comprehensively selecting groundtruth instance pairs, in Table 5.3, we also compare it to state-of-the-art algorithms with the frequently adopted metrics for evaluating candidate selection systems as defined in Equations 5.6 and 5.7.

$$PC = \frac{|true\ matches\ in\ candidate\ set|}{|true\ matches|} \tag{5.6}$$

$$RR = 1 - \frac{|candidate\ set|}{N * M} \tag{5.7}$$

The results show that our proposed on-the-fly candidate selection algorithm was not able to cover as sufficient groundtruth pairs as the the other systems did (as indicated by the metric PC). Although these missing groundtruth pairs didn't affect the final coreference results significantly, when combined with other actual entity coreference systems that are able to detect those missing pairs, there is still the risk that the final coreference F1-score be impacted substantially. We will introduce in Chapter 6 a candidate pre-selection technique that does not only provide comparably good coreference F1-score but also can achieve similar PC and RR to those state-of-the-art candidate selection systems.

5.4.5 Parameter Tuning

In this section, we would like to study the impact of the key parameters (as listed in Table 5.1) in our on-the-fly candidate selection algorithm.

Examining the Impact of Different Thresholding Approaches. First of all, we examine the effectiveness of our proposed pruning techniques by comparing to their alternatives. We compare different ways to adjust the threshold on instances' matching history similarity (Section 5.2): *Fixed* (using a fixed threshold), *Bump* (starting with 0 and then bumping to a fixed value), *Sigmoid* (σ=0 in Eq. 5.3) and *M-Sigmoid* (σ=30% in Eq. 5.3). We also consider removing our context pruning strategy (*M-Sigmoid\Eval*) to study its impact.

Table 5.4: Feature Evaluation. *Precision*, *Recall* and *F1-score* are the relevant measures for the actual entity coreference phase; *Time*: the runtime for the entire process, including both candidate selection and entity coreference.

Dataset	System	*Precision* (%)	*Recall* (%)	*F1* (%)	*Time* (s)
RKB Person	M-Sigmoid	95.02	**89.52**	**92.18**	7.66
	Sigmoid	95.05	89.22	92.04	7.56
	Bump	95.14	88.76	91.83	7.51
	Fixed	95.18	88.57	91.75	**7.26**
	M-Sigmoid\Eval	**95.23**	89.28	92.15	14.48
RKB Publication	M-Sigmoid	99.66	**98.10**	**98.87**	23.06
	Sigmoid	99.66	98.03	98.84	22.70
	Bump	99.66	97.91	98.78	22.66
	Fixed	**99.66**	97.80	98.72	**20.86**
	M-Sigmoid\Eval	99.49	98.09	98.79	51.00
SWAT Person	M-Sigmoid	99.49	**90.88**	**94.99**	8.61
	Sigmoid	99.49	**90.88**	**94.99**	8.58
	Bump	99.49	**90.88**	**94.99**	8.32
	Fixed	**99.51**	90.59	94.83	**8.27**
	M-Sigmoid\Eval	99.49	**90.88**	**94.99**	19.78

In Table 5.4, comparing different thresholding methods, *M-Sigmoid* has better recall than its alternatives; with similar precision to others, it achieves the best F1-scores. *Fixed* runs the fastest by using the upper bound threshold along the whole process. Although

the F1 results appear to be numerically close, they are in fact statistically significant. A two-tailed t-test on the F1-scores of applying *M-Sigmoid* and its thresholding alternatives to the 10 10K non-overlapping testing sets shows (10 F1-scores for each of the compared alternatives): for RKB Person, the differences are statistically significant with P values of 0.0001; for RKB Publication, the differences between *M-Sigmoid* and *Fixed*, *Sigmoid* and *Bump* are significant with P values of 0.0002, 0.0009 and 0.0001. Furthermore, *M-Sigmoid* is about 1.8 to 2 times faster than *M-Sigmoid*$\backslash$*Eval*, showing the effectiveness of our evaluation function for context pruning.

Varying the Percentage of Context Used for Filtering (k in Table 5.1). In our on-the-fly pruning based entity coreference algorithm (Algorithm 6), at line 6, after a pair of instances pass the history similarity check, we compute their similarity based upon their top k percent most important context (sorted based on path weight). In previous experiments, we set k to be 5, i.e., we only use the top 5% context for this purpose. Here, we test our algorithm on different k values and examine if by adopting more context information, we could achieve higher recall and F1-score without slowing down the coreference process too much.

First of all, we show the runtimes of different k values in Figure 5.2. As expected, when we use more context information, the runtime of the coreference process takes longer. This is more obvious for RKB Publication than for the other two person datasets, since an RKB Publication instance has more paths than person instances have. Generally speaking, when expanded to depth 2, a typical pattern for person instances is "person has publication, first name, last name and publication has author, citation, title, date, proceeding/journal, web_address, etc.". Because we stop the expansion process at depth 2, "proceedings and journals" will not be further expanded. While for publication instances, we normally have

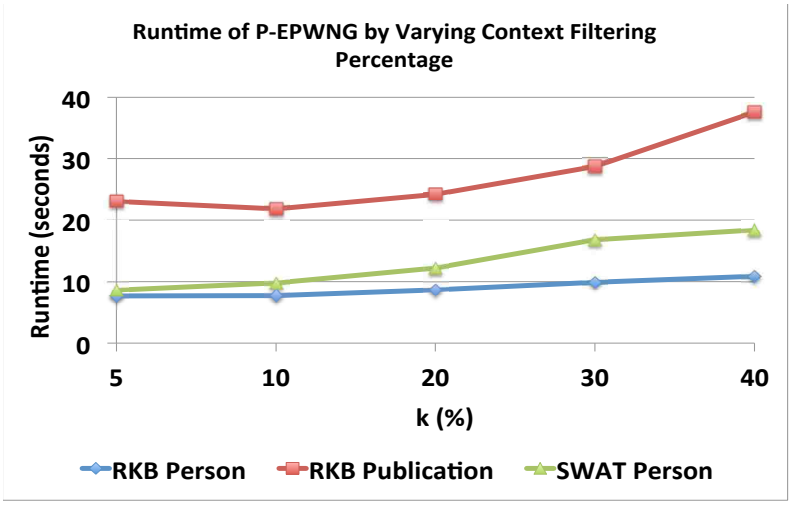


Figure 5.2: Runtime of Tuning the Number of Paths (k)

"publication has author, citation, title, date, citation, proceeding/journal, web_address; author has publication, first name, last name; proceeding/journal has publication_date, issue_number, etc; and citation is cited by other publications". Here, first of all, at depth 2, "proceedings and journals" will be expanded to get more information. Furthermore, a paper typically cites many other papers and could be cited many times at the same time; essentially, for publication instances, we have two level of citation networks. Overall, these cause publication instances to typically have much more number of paths than person instances would have. In our datasets, on average, an instance has 31.97, 41.69 and 162.63 paths as its context in RKB Person, SWAT Person and RKB Publication respectively.

As a second part of the experiment, we show the precision, recall and F1-score on all the three datasets in Table 5.5. It is interesting to see that we are not always getting higher recall and F1-score as we use more and more context information. Given two instances a and b, as we can see from line 6 to 10 in Algorithm 6, the similarity calculated with the k% context will be used to determine if instance a should be added to the history of instance b; and such history information is then used for initial filtering when considering

Table 5.5: Testing the Impact of the Percentage of Context k Used for Filtering. We bold the best scores of each metric among all tested k values.

Dataset	Metric	k (%)				
		5	10	20	30	40
RKB Person	Precision	**95.02**	95.02	94.89	94.87	94.81
	Recall	89.52	**89.72**	89.69	89.39	89.16
	F1-Score	92.18	**92.25**	92.21	92.04	91.9
SWAT Person	Precision	**99.49**	99.48	**99.49**	**99.49**	**99.49**
	Recall	90.88	**90.9**	90.88	90.87	90.87
	F1-Score	**94.99**	**94.99**	**94.99**	94.98	94.98
RKB Publication	Precision	**99.66**	99.66	99.65	99.65	99.65
	Recall	**98.1**	98.08	98.05	98.02	98.02
	F1-Score	**98.87**	98.86	98.84	98.83	98.83

other instances placed after instance a (as determined by our Lucene-based unsupervised sorting mechanism). Because our context information (paths) are sorted based on their path weight, as we employ more context, we are actually starting to incorporate more and more low-weight paths. Our coreference algorithm could then potentially be overwhelmed by such low-weight information and therefore generate more inaccurate matches. Such inaccurate matches could further pollute the history information of the instances, and finally cause the histories of two coreferent instances to have insufficient overlap, i.e., the Jaccard similarity between instances' matching histories falls below threshold. For RKB Person and SWAT Person, the best F1-scores are achieved when k=10%. The best F1-score for RKB Publication happens at k=5%; and this is because for publication instances, the top 5% context includes publication titles which typically are the most useful information for a coreference algorithm to disambiguate different instances.

Testing the Context Pruning Function. Another parameter in our *P-EPWNG* algorithm is γ, the maximum tolerance level for context pruning as described in Section 5.3. In this experiment, we vary the value of this parameter to examine its impact. The higher γ we use, the more context information (paths) we will use for the actual coreference.

Figure 5.3 shows the runtime by employing different γ values. Here, *All* means that we

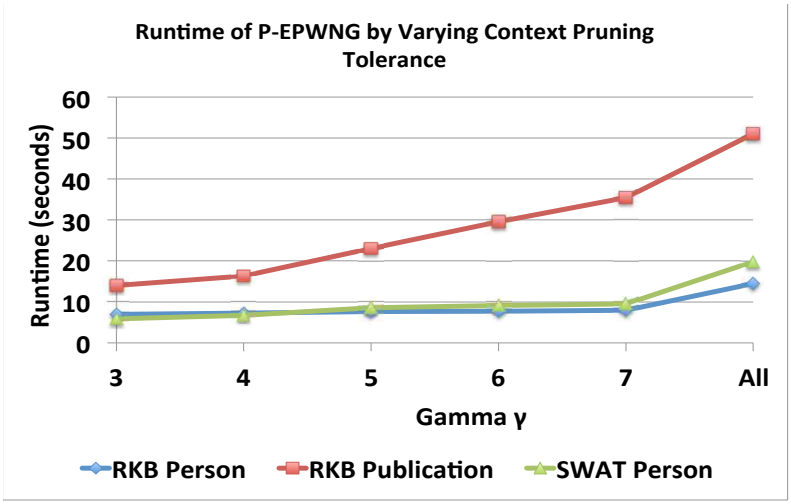


Figure 5.3: Runtime of Tuning Context Pruning Tolerance (γ)

do not use our context pruning, i.e., our coreference algorithm will exhaust all the available paths. As expected, by using more and more paths, we notice longer runtime. However, because we still utilized our history-based filtering, even when we used all the available context information, the runtime didn't increase too dramatically.

Before seeing the actual measured values of precision, recall and F1-score, one hypothesis that we may make is that as we make the cut earlier (using a lower tolerance level), we would expect a lower recall. However, as demonstrated in Table 5.6, as we begin to tolerate more and more context of potentially lesser value, not only does it result in longer runtime, but we are not always getting better recall and F1-score in return.

On the RKB Publication dataset, we typically get higher recall as we adopt a higher γ value, i.e., allowing the algorithm to consider more paths that may not contribute significantly to the final similarity measure. However, in contradiction to our hypothesis, on the RKB Person dataset, we are actually getting lower recall as we use more context information. To be able to explain this interesting situation, let's examine the following concrete

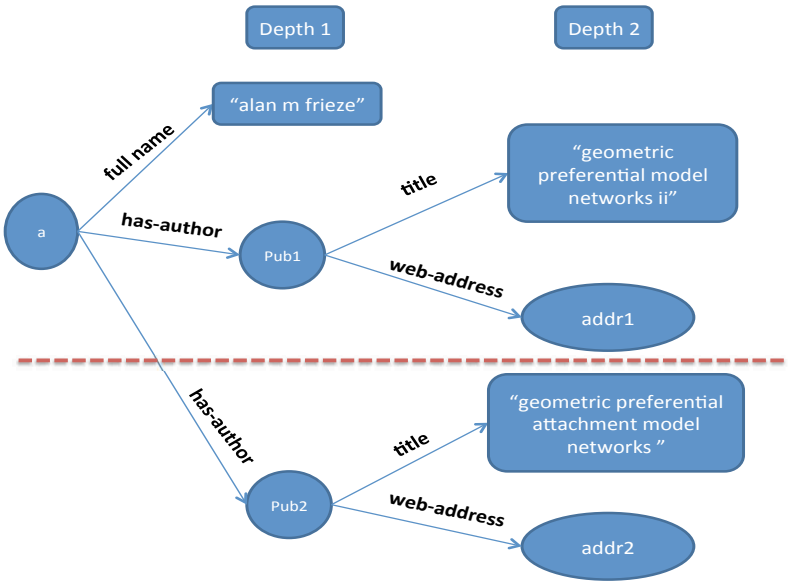
Table 5.6: Testing the Impact of the Number of Paths γ Used for Actual Entity Coreference. We bold the best scores of each metric among all tested γ values.

Dataset	Metric	γ					
		3	4	5	6	7	All
RKB Person	Precision	94.89	94.97	95.02	95.07	95.1	**95.23**
	Recall	**89.61**	89.56	89.52	89.49	89.48	89.28
	F1-Score	92.16	92.18	92.18	92.18	**92.19**	92.15
SWAT Person	Precision	99.35	99.46	**99.49**	**99.49**	**99.49**	**99.49**
	Recall	**90.9**	90.88	90.88	90.88	90.88	90.88
	F1-Score	94.93	94.97	**94.99**	**94.99**	**94.99**	**94.99**
RKB Publication	Precision	99.62	**99.66**	**99.66**	**99.66**	**99.66**	99.49
	Recall	98.05	98.06	**98.1**	**98.1**	**98.1**	98.09
	F1-Score	98.83	98.85	**98.87**	**98.87**	**98.87**	98.79

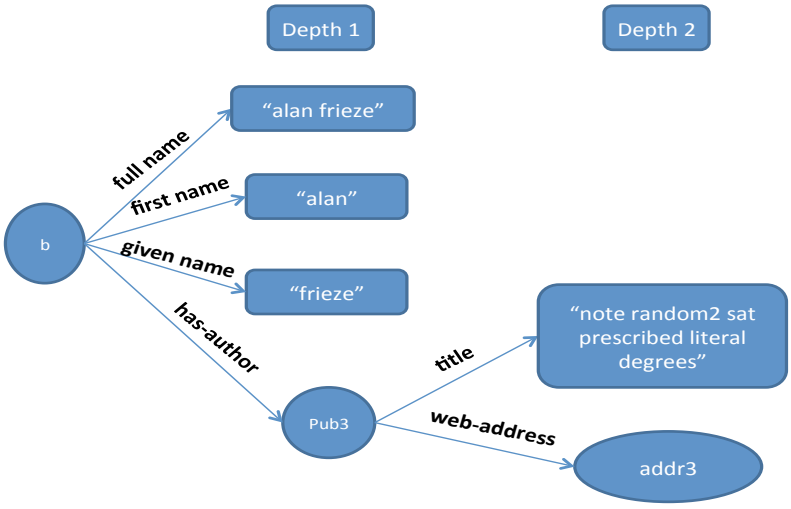
example.

We have two instances identified as a and b, and we show their expanded neighborhood graphs (their context) in Figures 5.4(a) and 5.4(b) respectively. These two instances are coreferent according to the groundtruth. First of all, we can see that a and b share highly similar full names, which is a very good indicator for them to be identified as coreferent. However, the two instances actually come from two different data sources: instance a is from DBLP, while instance b is from EPrints; and it is possible that instances from different data sources do not share sufficient publication information, which is exactly the case in the given example.

When we set γ to be 3, the paths above the red dashed line in Figure 5.4(a) will be considered for detecting coreference. With these paths, we will get a good match on their full names; while in the mean time, the similarity score will also be penalized by having mismatches on their depth-two paths, i.e., titles and web addresses. Fortunately, their similarity score can still exceed our threshold and thus they are treated as coreferent. However, if we set γ to be 4, the paths below the red dashed line will also be considered. In this case, we have more penalties by having more mismatches on titles and web addresses,



(a) Neighborhood Graph of Instance a



(b) Neighborhood Graph of Instance b

Figure 5.4: An Example of Having Lower Recall by Utilizing More Context Information

which finally causes the similarity of a and b to fall under the threshold. The drawback of applying penalties on both mismatched titles and web addresses is to cause the system to lose one pair of true match; however, this could potentially help to filter out a pair of non-coreferent instances that happen to share highly similar names.

Recall the way we calculate path score. We first check if the predicates at corresponding depth of two paths are comparable; we then compute the similarity between the end

nodes of comparable paths. The depth-two paths between the two instances shown above certainly satisfy our path comparability condition; while the problem is really that: *Are Pub1/Pub2 and Pub3, the publications represented by different identifiers, coreferent?*. If *Pub1* is coreferent with *Pub3*, then we should certainly apply the penalty; otherwise not. This actually comes down to the question that whether we should take into account those intermediate nodes, in addition to predicates, when we determine path comparability?

In order to perform this additional checking, we should firstly know whether two intermediate nodes represent the same entity; however, we may not have such knowledge when given a "brand new" dataset. Furthermore, as we have shown in a previous experiment (Table 3.1 in Section 3.4.4), by simply checking if two intermediate nodes are syntactically identical, we suffer from having much lower recall. One interesting future work would be to adopt some kind of bootstrapping strategy to perform iterative entity coreference. Initially, due to in-comprehensive coreference knowledge, we check intermediate nodes by only looking at their syntactic form, which may enable the system to achieve high precision but low recall. As we are obtaining more coreference information, in later iterations, we can then apply penalties to real mismatches (those caused by having coreferent intermediate nodes but distinct objects) to reduce the impact on recall.

When Should We Start Introducing the History-based Pruning. Another key parameter in our system is σ, indicating when we start applying our history-based pruning. In our previous experiment, we set σ to be 30%, meaning we start doing history-based filtering after 30% of the estimated groundtruth pairs have been covered by the processed instances. It would be natural to assume that as we adopt lower γ values, we would expect lower recall, since we start the filtering early on in the coreference process.

Table 5.7 shows the results of applying different σ values, ranging from 10% to 50%.

Here, we do not see any significant difference on runtime for different thresholds. The reason

Table 5.7: Testing the Impact of Starting Utilizing History-based Filtering at Different Estimated Groundtruth Coverage. We bold the best scores of each metric among all tested σ values.

Dataset	Metric	σ (%)				
		10	20	30	40	50
RKB Person	Precision	**95.04**	95.03	95.02	94.96	94.93
	Recall	89.32	89.43	89.52	89.63	**89.8**
	F1-Score	92.08	92.14	92.18	92.21	**92.29**
	Runtime (s)	**7.22**	7.57	7.66	8.17	8.15
RKB Publication	Precision	99.66	99.66	99.66	99.66	**99.66**
	Recall	98.04	98.06	98.1	98.17	**98.28**
	F1-Score	98.84	98.85	98.87	98.91	**98.97**
	Runtime (s)	**22.81**	22.87	23.06	23.22	23.76
SWAT Person	Precision	99.49	99.49	99.49	99.49	**99.49**
	Recall	90.88	90.88	90.88	90.88	**90.88**
	F1-Score	94.99	94.99	94.99	94.99	**94.99**
	Runtime (s)	**8.11**	8.65	8.61	8.62	8.62

is that even when we ignore history-based filtering for the first σ percent of the estimated groundtruth, we still have other effective filtering components, including *Share_A_Token* and top k% context-based filtering. If the history-based filtering is ignored for a pair of instances (the *Thresholding* function at Line 3 of algorithm 6), we only compute their similarity based on their top 5% weighted context, which is relatively fast. Therefore, we do not see much difference on runtime here.

If we look at the precision, recall and F1-score of different σ values, the results are generally consistent with our hypothesis. On the two RKB datasets, when we increase the value of σ, we start to have higher recall. Higher σ values mean that we apply history-based filtering at a later stage, thus fewer groundtruth instance pairs were falsely filtered out. On the other hand, recall that one of the positive aspects of applying candidate selection techniques is to be able to remove some of the potential false positives. As we adopt higher σ values, we do filtering less aggressively; as a consequence, our system becomes less capable of

filtering out false positives, which leads to the gradual decline of precision on RKB Person. Varying this threshold did not impact the performance on SWAT Person, except that it took a little longer as we adopt σ values higher than 10%.

Configuring the Upper bound of the History-based Filtering. In our modified Sigmoid function (Equation 5.3), we use the parameter K to control the upper bound of the history-based filtering. K determines the highest value that the *Thresholding* function at Line 3 of Algorithm 6 would return. The higher K we adopt, the more aggressively we would perform filtering.

In Table 5.8, we show the results of different K values, ranging from 0.1 to 0.5. As

Table 5.8: Varying the Filtering Upper bound of the Modified Sigmoid Function. We bold the best scores of each metric among all tested K values.

Dataset	Metric	K				
		0.1	0.2	0.3	0.4	0.5
RKB Person	Precision	94.97	95.02	95.07	**95.09**	**95.09**
	Recall	**89.74**	89.52	89.32	89.17	88.99
	F1-Score	**92.27**	92.18	92.09	92.03	91.93
	Runtime (s)	8.52	7.66	7.25	7.03	**6.86**
RKB Publication	Precision	**99.66**	**99.66**	**99.66**	**99.66**	**99.66**
	Recall	**98.24**	98.1	97.98	97.9	97.86
	F1-Score	**98.94**	98.87	98.81	98.77	98.75
	Runtime (s)	23.7	23.06	22.08	22.41	**21.86**
SWAT Person	Precision	99.49	99.49	99.49	**99.81**	**99.81**
	Recall	**90.88**	**90.88**	90.87	90.34	90.32
	F1-Score	**94.99**	**94.99**	94.98	94.83	94.82
	Runtime (s)	8.93	8.65	8.58	**8.52**	**8.52**

we are applying higher upper bounds, we are performing filtering more aggressively and will compute the top k% similarity for fewer pairs, therefore the length of the process is shortened. Furthermore, the trends of precision, recall and F1-score of different K values look intuitive. On all datasets, as we adopt higher K values, we typically achieve lower recall; in some situations, we also achieve higher precision, due to the fact that candidate selection helps to filter out some of the potential positives.

Stripping Off the *Share_A_Token* and *History_Sim* Module. A final test we would like to perform is to study the impact of the *Share_A_Token* and *History_Sim* modules. In this experiment, we compare to two alternatives: each alternative drops one of the two components. *m_sigmoid* uses all of our proposed filtering techniques; *m_sigmoid\share_a_token* is a system where we do not use the *Share_A_Token* function at Line 1 of Algorithm 6; *m_sigmoid\history_sim* is another system where we drop our history-based filtering module, Line 3 and 4 in Algorithm 6.

First of all, we present the runtime comparison of the three systems in Figure 5.5. We

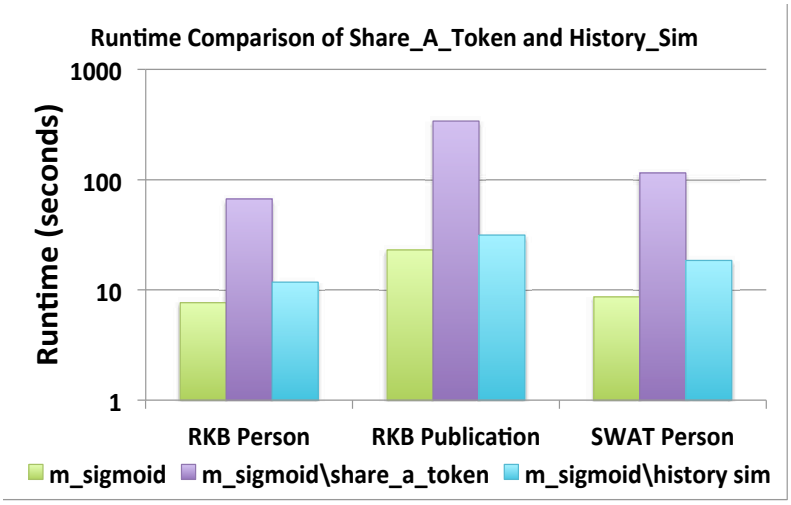
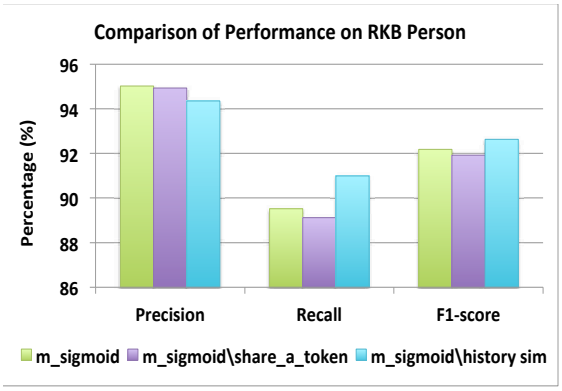


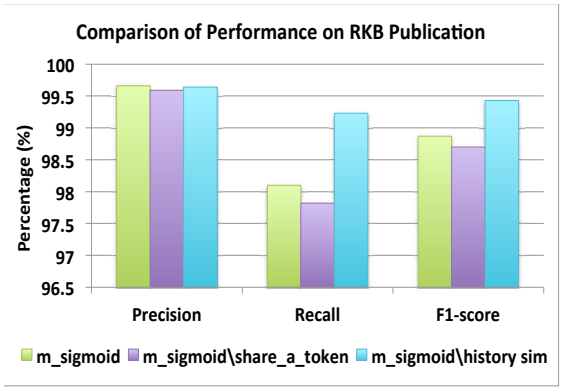
Figure 5.5: Runtime Comparison of Dropping *Share_A_Token* and *History_Sim*

use a logarithmic scale with base 10 for the y-axis. We can see that both modules actually have significant impact on the entire entity coreference process. Without using the *Share_A_Token* function, the process was slowed down by a factor of 9, 8 and 15 on RKB Person, SWAT Person, and RKB Publication respectively. Although dropping the *History_Sim* component had less impact on runtime, the *m_sigmoid* system was 35%, 53% and 26% faster than *m_sigmoid\history_sim* on RKB Person, SWAT Person and RKB Publication respectively, still showing its effectiveness in speeding up the coreference process.

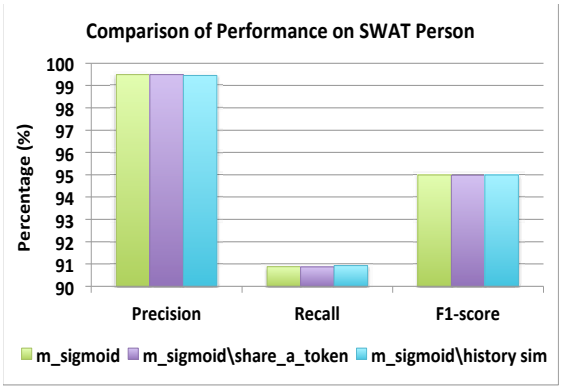
We also compare the precision, recall and F1-score of the three systems in Figures 5.6(a) to 5.6(c). On one hand, *m_sigmoid* achieves the highest precision, since perform-



(a) RKB Person



(b) RKB Publication



(c) SWAT Person

Figure 5.6: Impact of *Share_A_Token* and *History_Sim* on Precision, Recall and F1-score

ing filtering more aggressively could help to filter out some of the potential false positives. The improvement of precision on SWAT Person is not quite obvious: *m_sigmoid* and *m_sigmoid\share_a_token* achieve the same precision of 99.49% while *m_sigmoid\history_sim* has a precision of 99.45%.

On the other hand, we can see that *m_sigmoid\history_sim* has better recall than *m_sigmoid*, which indicates that our history-based approach is really an approximate or imprecise way of filtering un-coreferent instance pairs; and this is consistent with the results in Table 5.3, where our system did not achieve as good Pairwise Completeness for the

candidate selection phase as the other state-of-the-art systems did. However, given the fact that *m_sigmoid* did not sacrifice too much groundtruth pairs, it was still able to get decent F1-score in general.

Compared to *m_sigmoid*, *m_sigmoid\share_a_token* actually achieved a little lower recall. Presumably, we would not expect *Share_A_Token* to have any impact on recall, since we will just be wasting some additional effort in computing instance pairs that do not even share a single token in their top k% context (Line 6 of Algorithm 6). However, our *Share_A_Token* module also has an additional Lucene-based filtering sub-component (described in the second paragraph of Section 5.1). We only let two instances pass this check if they share at least one common token and their top k% contexts are similar to a certain degree. Such similarity scores are computed as part of the ranking processing of Lucene.

Let's examine why *m_sigmoid\share_a_token* achieves lower recall with the following concrete example. Suppose we have a pair of coreferent instances: 46986 and 1657757 (we use integers to identify instances in our database). When we do use *Share_A_Token*, the histories of both instances only include instance 85970; thus, they pass the history check and are deemed to be coreferent by our algorithm. However, without *Share_A_Token*, the history of 1657757 includes many noisy instances, one of which is 80522. As a consequence, 46986 and 1657757 did not have sufficiently overlapping histories and were filtered out.

Here, in Figure 5.7, we show the top 5% context of 1657757 and 80522 to explain why 80522 was included in the history of 1657757. Recall that in our coreference algorithm, given a path, we pick the highest similarity it could have with all comparable paths of another instance. In this example, for the *full-name* path of 80522, we compute its similarity to all three comparable *full-name* paths of 1657757; and the highest similarity happens between "K Tanaka" and "K Datta", because they share a common small token "K" as part of

```
Turtle:
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://www.example.org/person#1657757> foaf:name "Amitava Datta" .
<http://www.example.org/person#1657757> foaf:name "K Datta" .
<http://www.example.org/person#1657757> foaf:name "Aman Datta" .

<http://www.example.org/person#80522> foaf:name "K Tanaka" .
```

Figure 5.7: Top 5% Context of Instances 1657757 and 80522

their full names (we used Jaccard similarity for string matching). Therefore, our top k% similarity calculation function considers 80522 to be similar to 1657757 and thus is added to the history of 1657757. With *Share_A_Token*, these two instances will not be able to pass the *Share_A_Token* check. Even though they share one small token "K", the Lucene similarity score of their top k% context is not sufficiently high.

Through our parameter analysis, it is recommended that we use just enough context (k, Table 5.5) for filtering. Using too much may not necessarily improve recall, since context of lesser value will be included. Depending on the actual application scenarios, we may start performing our history-based filtering (σ, Table 5.7) sooner or later. When runtime is really critical, we could start the filtering early by sacrificing a reasonable level of recall; otherwise, higher values should be employed to ensure better coverage on coreferent instance pairs. Situated similarly, a lower upper bound (K, Table 5.8) for history-based filtering should be used in order to not miss too many true matches but noticeable runtime improvement can be achieved by adopting higher values for this parameter. Finally, our *Share_A_Token* and *History_Sim* components (Figures 5.6(a) to 5.6(c)) have been verified to be very effective in scaling the system to large scale datasets, therefore both of them should be retained, even though *History_Sim* may falsely filter out true matches due to noisy matching histories.

Chapter 6

Scaling Entity Coreference Using Domain-Independent Candidate Selection

One major drawback of the on-the-fly algorithm is that it does miss a non-trial amount of groundtruth pairs during its filtering process. For entity coreference systems that can actually detect those missing pairs, their performance could be significantly impacted. In order to achieve both good coreference F1-score and decent candidate selection coverage on true matches, in this chapter, we present another candidate selection algorithm that pre-selects candidate pairs by only comparing their most disambiguating partial context with a lightweight method to speed up the entire coreference process.

Several interesting questions then arise. First, manually choosing the information to compare might not work for all domains due to limited availability of domain expertise. Also, candidate selection algorithms should cover as many true matches as possible and reduce many true negatives. Algorithms that only try to maximize one aspect are not ideal.

Finally, the candidate selection algorithm itself should scale to very large datasets.

The candidate selection algorithm to be presented in this chapter possesses the properties discussed above [21]. Although our algorithm is designed for RDF data, it generalizes to any structured dataset. Given an RDF graph and the types of instances to do entity coreference on, we discover a set of datatype properties as the key for candidate selection that both discriminate and cover the instances well in an unsupervised and domain-independent manner. In order to support efficient look-up for similar instances, we index the instances on the discovered predicates' object values and adopt a character level n-gram based string similarity measure to select candidate pairs. We evaluate this algorithm on the same three instance categories from the RKB and SWAT datasets and another three structured datasets frequently used for evaluating entity coreference systems. In addition to using the traditional metrics: Pairwise Completeness, Reduction Ratio and F_{cs}, we perform the two-phase evaluation and compare to state-of-the-art systems on the overall runtime and the final coreference F1-score.

6.1 Iterative Candidate Selection Key Discovery

As discussed, candidate selection is the process of efficiently selecting instance pairs that are possibly coreferent by only comparing part of their context information. Therefore, the information we will compare needs to be disambiguating enough to the instances. For example, a person instance may have the following triples as shown in Figure 6.1:

Intuitively, we might say that last name could disambiguate this instance from others better than first name which is better than the place where he lives in. The reason could be that the last name *Henderson* is less common than the first name *James*; and a lot more people live in the United States than those using *James* as first name. Therefore, for

```
Turtle:
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.geonames.org/> .

<http://www.example.org/person#100> foaf:lastname "Henderson" .
<http://www.example.org/person#100> foaf:firstname "James" .
<http://www.example.org/person#100> geo:livesin "United States" .
```

Figure 6.1: Sample Triples of a Person Instance

person instances, we might choose the object values of *has-last-name* for candidate selection. However, we need to be able to automatically learn such disambiguating predicate(s) in a domain-independent manner. Furthermore, the object values of a single predicate may not be sufficiently disambiguating to the instances. Take the above example again, it could be more disambiguating if we use both last name and first name.

Algorithm 7 presents the process for learning the candidate selection key, a set of datatype predicates, whose object values are then utilized for candidate selection. Triples with datatype predicates use literal values as objects. The goal is to iteratively discover a set of predicates (the candidate selection key) whose values are sufficiently discriminating (discriminability) such that the vast majority of instances in a given dataset use at least one of the learned predicates (coverage). The algorithm starts with an RDF graph G (a set of triples, <i,p,o>) and it extracts all the datatype predicates (key_set) and the instances (I_C) of certain categories (C) (e.g., person, publication, etc.) from G. Then, for each predicate $key \in key\_set$, the algorithm retrieves all the object values of the key for instances in I_C. Next, it computes three metrics: discriminability, coverage as shown in Equations 6.1 and 6.2 respectively and a F1-score (F_L) on them.

$$dis(key, I_C, G) = \frac{|\{o|t =< i, key, o >\in G \wedge i \in I_C\}|}{|\{t|t =< i, key, o >\in G \wedge i \in I_C\}|} \tag{6.1}$$

Algorithm 7 Learn_Key(G, C), G is an RDF graph, consisting a set of triples, C is a set of instance types

1. $key\_set \leftarrow a\ set\ of\ datatype\ predicates\ in\ G$
2. $I_C \leftarrow \{i| < i$,rdf:type$, c >\in G \wedge c \in C\}$
3. $satisfied \leftarrow$ **false**
4. **while not** $satisfied$ **and** $key\_set \neq \emptyset$ **do**
5. **for** $key \in key\_set$ **do**
6. $discriminability \leftarrow dis(key, I_C, G)$
7. **if** $discriminability < \beta$ **then**
8. $key\_set \leftarrow key\_set - key$
9. **else**
10. $coverage \leftarrow cov(key, I_C, G)$
11. $F_L \leftarrow \frac{2*discriminability*coverage}{discriminability+coverage}$
12. $score[key] \leftarrow F_L$
13. **if** $F_L > \alpha$ **then**
14. $satisfied \leftarrow$ **true**
15. **if not** $satisfied$ **then**
16. $dis\_key \leftarrow \arg\max_{key \in key\_set} dis(key, I_C, G)$
17. $key\_set \leftarrow combine\ dis\_key\ with\ all\ other\ keys$
18. $G \leftarrow update(I_C, key\_set, G)$
19. **return** $\arg\max_{key \in key\_set} score[key]$

$$cov(key, I_C, G) = \frac{|\{i|t =< i, key, o >\in G \wedge i \in I_C\}|}{|I_C|} \tag{6.2}$$

Note, i and o represent the subject and object of a triple respectively. If any predicate has a F_L higher than the given threshold α, the predicate with the highest F_L will be chosen to be the candidate selection key.

If none of the keys have a F_L above the threshold α, the algorithm combines the predicate that has the highest discriminability with every other predicate to form $|key\_set|$-1 virtual predicates, add them to key_set and remove the old ones. Furthermore, via the function $update(I_C, key\_set, G)$, for a new key, we concatenate the object values of different predicates in the key for the same instance to form new triples that use the combined virtual predicate as their predicate and the concatenated object values as their objects. These new triples and predicates are added to G. The same procedure can then be applied iteratively. In the learning process, we remove low-discriminability predicates. Because the

discriminability of a predicate is computed based upon the diversity of its object values, having low-discriminability means that many instances have the same object values on this predicate; therefore, when utilizing such object values to look up similar instances, we will not get a suitable reduction ratio.

For determining the key for candidate selection, we re-use the proposed predicate discriminability in Section 3.2 and differences are: 1) We do not determine a candidate key only based on the discriminability of a predicate; instead, multiple predicates might be combined as the key; 2) We introduce two new metrics, *coverage* and F_{cs}, which are used in together with predicate discriminability in an unsupervised predicate discovery process to learn the candidate selection key.

During the candidate selection key discovery process, we remove predicates whose discriminability is lower than the threshold, β. On one hand, because the discriminability of a predicate is computed based upon the diversity of its object values, low discriminability means many instances have the same object values on this predicate; therefore, when trying to utilize such object values to look up similar instances (to be described in Section 6.2), more un-coreferent pairs could be selected. On the other hand, since such keys do not contribute much in disambiguating the instances, it will help to reduce the overall computational cost by removing them.

6.2 Indexing Ontology Instances for Efficient Lookup

With the learned predicates, for each instance, we present how to efficiently look up similar instances and compute their similarity based upon the object values of the learned predicates. One simple approach might be to conduct a pairwise comparison of the object values of the learned predicates for all pairs of instances, e.g., comparing last names and first

names for people instances. However, for a dataset with millions of instances, this simple method might not even scale itself. Therefore, we need a technique that enables efficient look-up for similar instances.

We adopt a traditional technique in the Information Retrieval (IR) research field, inverted index, to speed up the look-up process. Given an RDF graph G and the predicates PR learned by Algorithm 7, we build an index as shown in Figure 6.2. For a learned pred-

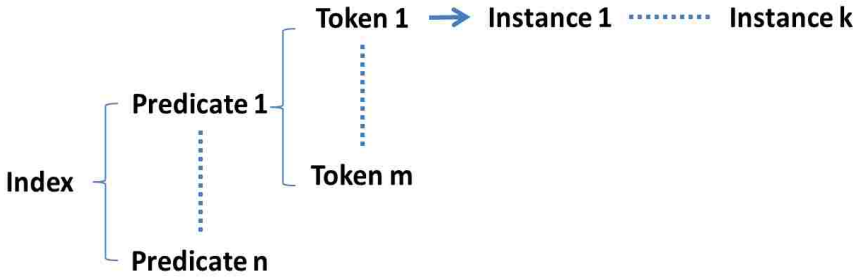


Figure 6.2: Index Structure

icate $p \in PR$, we extract tokens from the object values of triples using p; for each such token tk, we collect all instances that have triples with p and also at least one object value of such triples contains tk.

Figure 6.3 demonstrates a concrete example of how we convert RDF triples to the corresponding Information Retrieval style inverted index. Here, we have two ontology instances i

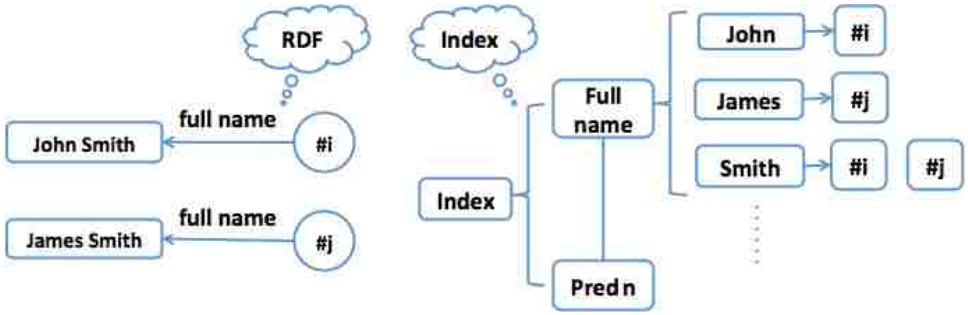


Figure 6.3: An Example of Converting RDF Triples to Information Retrieval Inverted Index

and j and let's assume only one predicate *full-name* was selected as the candidate selection

key. Both instances have a single triple for the *full-name* predicate with object values "John Smith" and "James Smith" respectively, which is shown on the left side of the diagram. Correspondingly, on the right side, we show the built inverted index. As we discussed before, for each selected predicate, we have one separate index. So, we see that we have the *Full name* index. This index has two major components: term list and posting list. Each token we extract from the object values of the *full-name* predicate, "John", "James" and "Smith", is treated as a separate term; and each term is associated with a posting list that stores all instances whose full name contains this term. In this given example, we know that both instances i and j have the token "Smith" in their full name, thus they are both in the posting list of the term "Smith".

With the learned predicates and built inverted indices, each instance is associated with tuple(s) in the form of (instance, predicate, value). We define a function search(Idx, q, $pred$) that returns the set of instances for which the $pred$ field matches the boolean query q using inverted index Idx. Algorithm 8 presents our candidate selection process where t is a tuple and $t.v$, $t.p$ and $t.i$ return the value, predicate and instance of t respectively. For

Algorithm 8 Candidate_Selection(T,Idx), T is a set of tuples using predicates in the learned key; Idx is an inverted index

1. $candidates \leftarrow \emptyset$
2. **for all** $t \in T$ **do**
3. $\quad query \leftarrow the\ disjunction\ of\ tokens\ of\ t.v$
4. $\quad results \leftarrow \bigcup_{p \in Comparable(t.p)} search(Idx, query, p)$
5. $\quad$ **for all** $t' \in results$ **do**
6. $\quad\quad$ **if** $is\_sim(t, t')$ **then**
7. $\quad\quad\quad candidates \leftarrow candidates \cup (t.i, t'.i)$
8. **return** $candidates$

each tuple t, we issue a Boolean query, the disjunction of its tokenized values, to the index to search for tuples ($results$) with similar values on all predicates comparable to that of t. The search process performs an exact match on each query token. $is\_sim(t, t')$ returns true

if the similarity between two tuple values is higher than a threshold.

We look up instances on comparable fields. For example, in one of our datasets used for evaluation, we try to match person instances of both the *CiteSeer:Person* and the *DBLP:Person* classes where the key is the combination of *CiteSeer:Name* and *FOAF:Name*. So, for a tuple, we need to search for similar tuples on both predicates. Assuming we have an alignment ontology where mappings between classes and predicates are provided, two predicates p and q are comparable if the ontology entails $p \sqsubseteq q$ (or vice versa).

6.3 Refining Index Lookup Results

To further reduce the size of the candidate set, it would be necessary to adopt a second level similarity measure between a given instance (i) and its returned similar instances from the Boolean query. Otherwise, any instance that shares a token with i will be returned. In this paper, we compare three different definitions of the function is_sim. The first one is to directly compare (*direct_comp*) two tuple values (e.g., person names) as shown in Equation 6.3.

$$String\_Matching(t.v, t'.v) > \delta \tag{6.3}$$

where t and t' are two tuples; $String\_Matching$ computes the similarity (e.g., Jaccard) between two strings. If the score is higher than the threshold δ, this pair of instances will be added to the candidate set. Although this might give a good pairwise completeness by setting δ to be low, it could select a lot of non-coreference pairs. One example is person names. Person names can be expressed in different forms: first name + last name; first initial + last name, etc.; thus, adopting a low δ could help to give a very good coverage on true matches; meanwhile, certain amount of non-coreferent pairs could also be included.

Another choice is to check the percentage of their shared highly similar tokens ($token\_sim$) as shown in Equation 6.4:

$$\frac{|sim\_token(t.v, t'.v)|}{\min(|token\_set(t.v)|, |token\_set(t'.v)|)} > \theta \tag{6.4}$$

where $token\_set$ returns the tokens of a string; sim_token is defined in Eq. 6.5:

$$\begin{aligned} &sim\_token(s_i, s_j) = \\ &\{token_i \in token\_set(s_i) | \exists token_j \in token\_set(s_j), \\ &String\_Matching(token_i, token_j) > \delta)\} \end{aligned} \tag{6.5}$$

where $s_{i/j}$ is a string and $token_{i/j}$ is a token from it. Without loss of generality, we assume that the number of tokens of s_i is no greater than that of s_j. The intuition behind Eq. 6.4 is that two coreferent instances could share many similar tokens, although the entire strings may not be sufficiently similar on their entirety. One potential problem with this similarity measure is that it may take longer to calculate because the selected literal values for some types of instances could be long (e.g., publication titles).

Instead of computing token level similarity, a third option is to check how many character level n-grams are shared between two tuple values as computed in Equation 6.6:

$$\frac{|gram\_set(n, t.v) \bigcap gram\_set(n, t'.v)|}{\min(|gram\_set(n, t.v)|, |gram\_set(n, t'.v)|)} > \theta \tag{6.6}$$

where $gram\_set(n, t.v)$ extracts the character level n-grams from a string. We hypothesize that the n-gram based similarity measure is the best choice. The intuition is that we

can achieve a good coverage on true matches to the Boolean query by examining the n-grams (which are finer grained than both tokens and entire strings) while at the same time effectively reducing the candidate set size by setting an appropriate threshold. We use *min* in the denominator for Equations 6.4 and 6.6 to reduce the chance of missing true matches due to missing tokens, spelling variations or misspellings (e.g., some tokens of people names can be missing or spelled differently).

6.4 Evaluation

6.4.1 Evaluation Datasets and Methods

The three RDF datasets that we used in previous chapters are also used for evaluating the proposed offline candidate selection algorithm. I randomly chose 100K instances for each instance category (RKB Person, RKB Publication and SWAT Person), split each 100K dataset into 10 non-overlapping and equal-sized subsets, run all algorithms on them and report the average. This enabled us to conduct the two-tailed t-test to test the statistical significance on the results of the 10 subsets from two systems. On average, there are 6,096, 4,743 and 684 coreferent pairs for each subset of RKB Person, RKB Publication and SWAT Person respectively.

Similar to the way we evaluate our on-the-fly candidate selection algorithm, I still adopt the two-phase approach here. In phase one, I use the three well adopted metrics PC, RR and F_{cs} from previous works [43, 40, 41] as discussed in Section 2.3.1. For phase two, the proposed entity coreference [16] (See Section 3) is used. I apply the proposed candidate selection technique on the RDF datasets discussed in Section 3.4 to select candidate pairs and run the proposed entity coreference algorithm on the candidate sets to measure the

F-score of the coreference phase and the runtime of the entire process, including candidate selection and coreference.

Table 6.1 shows the parameter setting for our evaluation. For the two key parameters α and β, we set them to be 0.9 and 0.2 respectively. When β is low, only a few predicates are removed for not being discriminating enough; when α is high, then I only select keys that discriminate well and are used by most of the instances. For *direct_comp* and *token_sim*, I varied δ from 0.1 to 0.9 and report the best results. I extract bigrams and compute Jaccard similarity for string matching in all experiments.

Table 6.1: Parameter Settings.

Parameter	Value	Description
α (Alg. 7)	0.9	Whether a key will be used for candidate selection
β (Alg. 7)	0.2	Threshold for filtering low-discriminability predicates
θ (Eqs. 6.4, 6.6)	0.8	If a pair of instances should be chosen as a candidate
δ (Eqs. 6.3, 6.5)	0.1 to 0.9	Threshold for string similarity on their entirety

6.4.2 Evaluation Results on RDF Datasets

From Algorithm 7, the discovered candidate selection key for each RDF dataset is as following:

RKB Person: full-name, job, email, web-addr and phone

RKB Publication: title

SWAT Person: CiteSeer:name and FOAF:name

For RKB Person, *full-name* has good coverage but is not sufficiently discriminating; while the other selected predicates have good discriminability but poor coverage. So, they were combined to be the key. For SWAT Person, neither of the two selected predicates has sufficient coverage; thus both were selected.

The proposed candidate selection method *bigram* (Eq. 6.6) is firstly compared to *direct_comp* (Eq. 6.3) and *token_sim* (Eq. 6.4) that use different string similarity measures. Also, we compare to our previous entity coreference algorithm *P-EPWNG* [20] that employs the on-the-fly filtering mechanism discussed in Chapter 5, and compare to the *EPWNG* [16] algorithm proposed in Chapter 3 that does not perform any candidate selection or context pruning, on the coreference results and runtime. Finally, we compare against the state-of-the-art candidate selection systems: *All-Pairs* [97], *PP-Join*(+) [98], *Ed-Join* [99], *FastJoin* [96], *IndexChunk* [95], and *BiTrieJoin* [94]. For coreference results, a system's best F-Score is reported by applying a threshold from 0.1 to 0.9.

Table 6.2 shows the results of different systems. We explain the meanings of the metrics here: $|Pairs|$: candidate set size; CST: time for candidate selection; RR: Reduction Ratio; PC: Pairwise Completeness; F_{cs}: the F1-score for RR and PC; F1: the F1-Score of Precision and Recall for the coreference results; Total: the runtime for the entire entity coreference process.

First of all, comparing within our own alternatives, in general, both *bigram* and *token_sim* has the best RR while *direct_comp* commonly has better PC. Our proposed candidate selection technique *bigram* selected almost as few pairs as *token_sim* on the two RKB datasets and even fewer pairs for SWAT Person, and always had better PC. Moreover, *bigram* enabled the entire coreference process to finish the fastest on all three datasets.

On RKB Person, for F_{cs}, *bigram* was not as good as *direct_comp* but is able to outperform all compared state-of-the-art systems. Statistically, the difference between *bigram* and *direct_comp* and *token_sim* is significant with P values of 0.0106 and 0.0004 respectively; the difference between *bigram* and all state-of-the-art systems is also statistically significant with a P value of 0.0001. Furthermore, by applying our entity coreference system

Table 6.2: Candidate Selection Results on RDF Datasets.

Dataset	System	*Candidate Selection*					*Coref*(%)	Time (s)
		$\|Pairs\|$	CST	$PC(\%)$	$RR(\%)$	$F_{cs}(\%)$	F1	$Total$
RKB Person	bigram (Eq. 6.6)	13,790	4.26	99.39	**99.97**	99.68	**93.14**	**5.80**
	direct_comp (Eq. 6.3)	72,080	4.12	99.56	99.86	**99.71**	92.86	9.02
	token_sim (Eq. 6.4)	**13,679**	4.65	97.68	**99.97**	98.81	92.91	6.57
	PEPWNG [16]	68,427	4.46	94.94	99.86	97.34	92.18	7.66
	EPWNG [16]	N/A	N/A	N/A	N/A	N/A	91.96	6,296.91
	Ed-Join [99]	207,643	1.31	99.60	99.58	99.59	92.84	63.31
	AllPairs [97]	454,972	**0.93**	**99.64**	99.09	99.36	92.52	83.76
	PPJoin+ [100]	454,972	1.02	**99.64**	99.09	99.36	92.52	82.96
	FastJoin [96]	443,846	3.92	**99.64**	99.11	99.38	92.52	81.75
RKB Publication	bigram (Eq. 6.6)	6,269	5.91	**99.99**	**99.99**	**99.99**	**99.76**	**8.03**
	direct_comp (Eq. 6.3)	49,639	5.73	99.98	99.90	99.94	99.71	18.32
	token_sim (Eq. 6.4)	**5,028**	7.66	99.79	**99.99**	99.89	99.69	9.59
	PEPWNG [16]	107,471	13.18	98.27	99.79	99.02	98.87	23.06
	EPWNG [16]	N/A	N/A	N/A	N/A	N/A	99.58	63200.71
	Ed-Join [99]	1,298,525	131.85	98.54	97.40	97.97	98.90	1330.20
	AllPairs [97]	581,005	1.41	99.21	98.84	99.02	99.27	340.14
	PPJoin+ [100]	581,005	**1.39**	99.21	98.84	99.02	99.27	342.21
	FastJoin [96]	583,839	75.78	99.22	98.83	99.03	99.27	430.61
SWAT Person	bigram (Eq. 6.6)	**8,058**	5.50	98.67	**99.98**	99.32	94.90	**6.78**
	direct_comp (Eq. 6.3)	90,092	4.83	99.78	99.82	**99.80**	94.95	17.61
	token_sim (Eq. 6.4)	17,562	7.22	96.79	99.96	98.35	94.96	9.52
	PEPWNG [16]	43,333	4.50	96.79	99.91	98.32	94.99	8.61
	EPWNG [16]	N/A	N/A	N/A	N/A	N/A	**94.99**	16968.00
	Ed-Join [99]	226,330	1.56	99.63	99.55	99.59	94.94	102.77
	AllPairs [97]	381,128	**0.79**	**99.80**	99.24	99.52	**94.99**	108.34
	PPJoin+ [100]	381,128	0.86	**99.80**	99.24	99.52	**94.99**	106.72
	FastJoin [96]	360,481	3.20	**99.80**	99.28	99.54	**94.99**	103.72

to the selected pairs, *bigram* has the best F1-score that is statistically significant compared to that of *direct_comp*, *All-Pairs/PP-Join(+)/FastJoin*, *EdJoin/IndexChunk*, and *BiTrieJoin/PartEnum* with P values of 0.0433, 0.0093, 0.0482 and 0.0001 respectively.

We observed similar results on SWAT Person. For the candidate selection F_{cs}, the difference between *bigram* and *direct_comp/token_sim/BiTrieJoin/PartEnum* and *EdJoin/All-Pairs/PP-Join(+)/FastJoin* is statistically significant with P values of 0.0001 and 0.0263 respectively. Similarly, *P-EPWNG* and *bigram* were able to run faster than all other compared systems. As for the $F1$-*score* of the actual coreference results, we did not observe any significant difference among the various systems.

On RKB Publication, *bigram* dominates the others in all aspects except for $|pairs|$ and candidate selection time. For F_{cs}, except for *direct_comp*, the difference between *bigram* and others is statistically significant with P values of 0.0001. As for the coreference results, although no statistical significance was observed between *bigram* and *direct_comp/token_sim*, statistically, *bigram* achieved a better F1-score than those state-of-the-art systems with a P value of 0.0001.

One thing to note is that on both RKB datasets, *token_sim* took longer to finish the entire process than *bigram* even with fewer selected pairs because it takes longer to select candidate pairs. It would potentially have to compare every pair of tokens from two strings, which was time-consuming. This was even more apparent on RKB Publication where titles generally have more tokens than people names.

Finally, we compare *bigram* to *EPWNG* [16], the algorithm that does pairwise comparison on all pairs of instances as introduced in Chapter 3. Table 6.2 shows that using *bigram* enables the entire process to run 2-3 orders of magnitude faster than *EPWNG*; and by applying candidate selection, the F-scores of the coreference results on both RKB

datasets did not drop and even noticeably better performance was achieved. For RKB Person and RKB Publication, the improvement on the coreference F-score is statistically significant with P values of 0.0001 and 0.0020 respectively. Such improvement comes from better precision: by only comparing the disambiguating information selected by Algorithm 7, candidate selection filtered out some false positives that could have been returned as coreferent by *EPWNG*. For instance, for RKB Person, one false positive could be detected by *EPWNG* by considering two people's publications that are accessible in the RDF graph by getting distant triples [16]; however, only by considering their most disambiguating information (e.g., name, job, etc.), they could be filtered out by candidate selection. In this case, candidate selection doesn't only help to scale the entire entity coreference process but also improves its overall F-score. This is consistent with what we observed for the on-the-fly pruning technique in Chapter 5.

6.4.3 Evaluation Results Using Standard Coreference Datasets

To show the generality of our proposed algorithm, we also evaluate it on three non-RDF but structured datasets frequently used for evaluating entity coreference algorithms: Restaurant, Hotel and Census as described earlier. We learned the candidate selection key for each dataset as following:

Restaurant: name

Hotel: name

Census: date-of-birth, surname and address_1

Here, we compare to five more systems: BSL [42], ASN [43], Marlin [92], AF [91] and Best Five [87] by referencing their published results. We were unable to obtain the executables

for these systems.

For this experiment, we apply candidate selection on each of the three full datasets. First, the scale of the datasets and their groundtruth is small. Also, each of the Restaurant and the Hotel datasets is actually composed of two subsets and the entity coreference task is to map records from one to the other; while for other datasets, we detect coreferent instances within all the instances of a dataset itself. So, it is difficult to split such datasets. We didn't apply any actual coreference systems to the candidate set here due to the small scale and the fact that we couldn't run some of the systems to collect the needed candidate sets. Instead, in order to accurately reflect the impact of RR, we suggest a new metric RR_{log} computed as $1 - \frac{\log |candidate\ set|}{\log (N*M)}$. In Table 6.2, on RKB Person, an order of magnitude difference in detected pairs between *bigram* and *Ed-Join* is only represented by less than 1 point in RR; however, a more significant difference in the total runtime was observed. With this new metric, *bigram* and *Ed-Join* have an RR_{log} of 46.13% and 32.77% respectively where the difference is now better represented by 13.36%. We also compute a corresponding F_{cs_log} using RR_{log}. For systems where we references reported results, we calculated $|pairs|$ from their reported RR; because BSL is supervised (thus the blocking was not done on the full dataset), we assumed the same RR as if it was done on the full dataset.

Table 6.3 shows the results. Since not all systems reported results on all datasets, we only report the available results here. Comparing within our own alternatives, for all datasets, *direct_comp* has the best PC; *bigram* and *token_sim* have identical RR, but *bigram* always has better PC. Furthermore, *bigram* always has the best F_{cs_log} and has better RR_{log} on Restaurant and Hotel but only slightly worse on Census than *token_sim*.

Table 6.3: Candidate Selection Results on Standard Coreference Datasets. *bigram*, *direct_comp* and *token_sim* refer to Equations 6.6, 6.3 and 6.4 respectively.

Dataset	System	*Candidate Selection*					
		$\|Pairs\|$	$RR(\%)$	$PC(\%)$	$F_{cs}(\%)$	$RR\_log(\%)$	$F_{cs\_log}(\%)$
Restaurant	bigram	**182**	**99.90**	98.21	99.05	**56.92**	**72.07**
	direct_comp	2,405	98.64	**100.00**	**99.31**	35.56	52.46
	token_sim	184	**99.90**	95.54	97.67	56.83	71.27
	All-Pairs [97]	1,967	98.89	99.11	99.00	37.22	54.12
	PP-Join [98]	1,967	98.89	99.11	99.00	37.22	54.12
	PP-Join+ [98]	1,967	98.89	99.11	99.00	37.22	54.12
	Ed-Join [99]	6,715	96.19	96.43	96.31	27.06	42.26
	BSL [42]	1,306	99.26	98.16	98.71	40.61	57.45
	ASN [43]	N/A	N/A	<96	<98	N/A	N/A
	Marlin [92]	78,773	55.35	**100.00**	71.26	6.67	12.51
Hotel	bigram	**4,142**	**97.21**	94.26	**95.71**	**30.06**	**45.58**
	direct_comp	10,036	93.24	96.69	94.94	22.63	36.67
	token_sim	4,149	**97.21**	90.56	93.77	30.04	45.12
	All-Pairs [97]	6,953	95.32	95.91	95.62	25.71	40.55
	PP-Join [98]	6,953	95.32	95.91	95.62	25.71	40.55
	PP-Join+ [98]	6,953	95.32	95.91	95.62	25.71	40.55
	Ed-Join [99]	17,623	88.13	98.93	93.22	17.90	30.31
	BSL [42]	27,383	81.56	**99.79**	89.76	14.20	24.86
Census	bigram	166,844	99.67	97.76	98.70	32.17	48.41
	direct_comp	738,945	98.52	98.08	98.30	23.77	38.27
	token_sim	163,207	99.67	96.36	97.99	32.30	48.38
	All-Pairs [97]	**5,231**	**99.99**	**100.00**	**99.99**	**51.70**	**68.16**
	PP-Join [98]	**5,231**	**99.99**	**100.00**	**99.99**	**51.70**	**68.16**
	PP-Join+ [98]	**5,231**	**99.99**	**100.00**	**99.99**	**51.70**	**68.16**
	Ed-Join [99]	11,010	99.98	99.50	99.74	47.50	64.30
	AF [91]	49,995	99.9	92.7	96.17	38.97	54.87
	BSL [42]	939,906	98.12	99.85	98.98	22.42	36.62
	Best Five [87]	239,976	99.52	99.16	99.34	30.12	46.21

Compared to other systems, on both Restaurant and Hotel, *bigram* has the best RR, F_{cs}, RR_{log} and F_{cs_log}, though its F_{cs_log} was only slightly better than that of $All\text{-}Pairs/PP\text{-}Join(+)$. Also, with better RR, it only has slightly worse PC than $All\text{-}Pairs/PP\text{-}Join(+)$/Marlin on Restaurant. Particularly, *bigram* has significantly better RR (15.65% and 9.08% higher) than BSL and $Ed\text{-}Join$ on Hotel; however it was not able to achieve a PC as good as these two systems did. If we consider larger datasets, such a significant difference in RR may save a great amount of runtime. Note that with the two new metrics, the impact of the number of selected pairs becomes more apparent, which we believe more accurately reflects

its impact. On Census, *All-Pairs/PP-Join*(+) achieved the best F_{cs} and F_{cs_log}; while *bigram* still achieved better RR than BSL and $BestFive$ with slightly worse PC. *bigram* only has a PC of 97.76% because our method only performs exact look-ups into the index; however, in this synthetic dataset, coreferent records were generated by modifying the original records, including adding misspellings, removing white spaces, etc. Therefore, some of the coreferent records could not even be looked up from the inverted index. In future work, we will explore techniques for efficient fuzzy retrieval to overcome this problem.

6.4.4 Applying Other Actual Entity Coreference Algorithms for Phase II Evaluation

As we noticed from Table 6.2, after applying our on-the-fly and offline candidate selection techniques, the *P-EPWNG* and *bigram* systems achieved higher precision for the entire coreference process; in the mean time, they were also able to maintain comparable recall to the other systems, thus enabling them to achieve satisfying coreference F1-scores in general. However, in the previous experiments, we used *EPWNG* [103], our own coreference algorithm, as the actual coreference algorithm in the second phase of two-phase evaluation process. In this section, we would like to explore more on this high-precision-low-recall phenomenon by applying another state-of-the-art actual coreference algorithm to examine the following questions: 1) If the other state-of-the-art systems can also be sped up significantly with our pruning technique; 2) If the other state-of-the-art entity coreference algorithms would also achieve higher precision after applying our candidate selection technique; 3) If the other systems would be able to maintain decent recall given that our pruning technique is bound to incorrectly filter out some of the groundtruth pairs.

For this experiment, we use another system *LogMap* [72, 73], as discussed in Section

2.3.4. And we compare the following four systems:

1. *LogMap* that has a built-in candidate selection module that indexes values of manually specified datatype properties and retrieves instances sharing at least one common token;

2. *LogMap_NoCS* is a modified version of *LogMap* with its own candidate selection module disabled;

3. *LogMap_On-The-Fly* uses our own on-the-fly candidate selection technique;

4. *LogMap_Offline* adopts the pre-selection technique proposed in this chapter.

For the latter two systems, we firstly run the on-the-fly and offline algorithms respectively to generate the candidate pairs and then apply *LogMap_NoCS* on the selected pairs to detect their coreference relationships.

Runtime Comparison

Figure 6.4 shows the runtime comparison of these four systems on RKB Person, SWAT Person, RKB Publication respectively. The y-axis represents runtime in seconds, and we use a logarithmic scale with base 10. First of all, after plugging in a candidate selection module, both *LogMap* and *LogMap_On-The-Fly* achieved significant runtime speedup compared to *LogMap_NoCS*. This verifies the need of adopting this additional filtering step in facilitating entity coreference on large-scale datasets. Furthermore, by adopting our own on-the-fly candidate selection algorithm, for RKB Person and SWAT Person, *LogMap_On-The-Fly* was able to run 25% and 60% faster than utilizing *LogMap*'s built-in candidate selection module, showing the effectiveness of our proposed algorithm. Although *LogMap_On-The-Fly* was not running as fast as *LogMap* on the RKB Publication dataset, it indeed achieves substantially

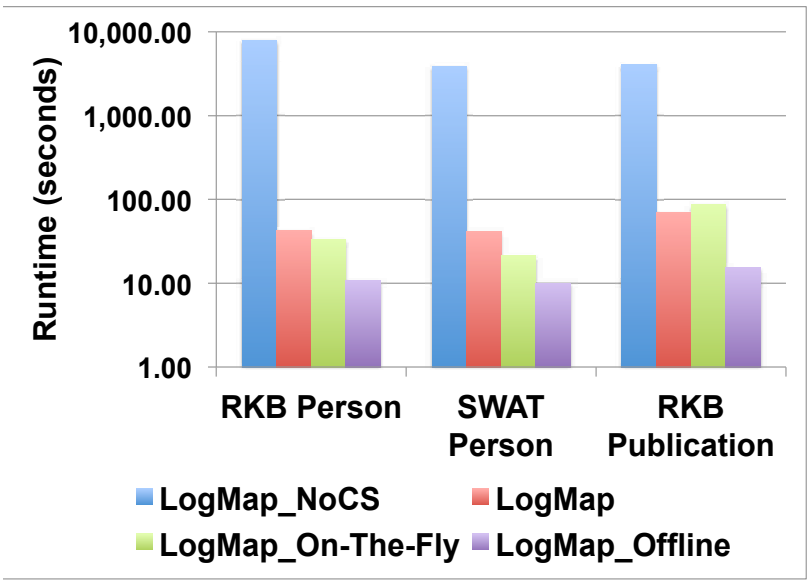


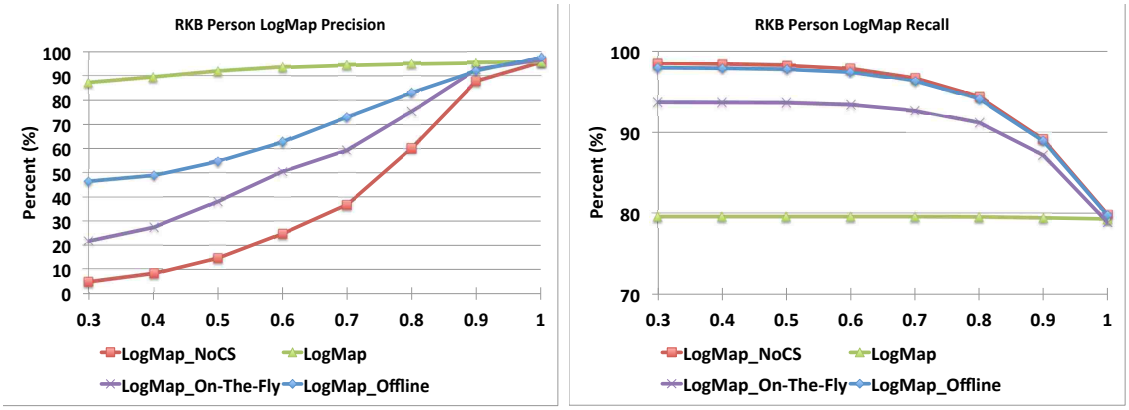
Figure 6.4: Runtime Comparison by Applying the LogMap Algorithm to Selected Candidate Instance Pairs

higher precision, recall and F1-score (to be presented in Figures 6.5(a) to 6.7(c)). Finally, when adopting our offline candidate selection technique, *LogMap_Offline* runs the fastest among all systems. And it is about 3, 2 and 6 times faster than *LogMap_On-The-Fly* on RKB Person, SWAT Person and RKB Publication respectively. This is consistent with the results in Table 6.2, where we see that the offline candidate selection algorithm selected remarkably fewer candidate pairs than the on-the-fly algorithm did.

Precision, Recall, and F1-Score

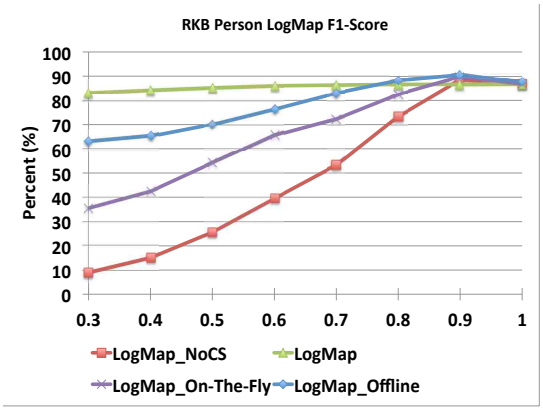
In addition to runtime, we want to verify the other two hypotheses we discussed above: whether other entity coreference systems can achieve higher precision and maintain decent recall by applying our candidate selection technique.

RKB Person. Figures 6.5(a) to Figures 6.5(c) demonstrate the precision, recall, and F1-score of the four systems on RKB Person. It is not surprising to see that *LogMap_NoCS* maintains the best recall for all thresholds, given that there is no candidate selection



(a) RKB Person Precision

(b) RKB Person Recall



(c) RKB Person F1-Score

Figure 6.5: Precision, Recall and F1-Score on RKB Person Dataset by Plugging Candidate Selection into LogMap

adopted. Due to the same fact, its precision is lower than the other two comparison systems; this shows that the utilizing candidate selection can effectively reduce the chance of having false positives. Although *LogMap_NoCS* could not achieve a decent F1-score at low thresholds, it was able to catch up with the other three systems at high thresholds as shown in Figure 6.5(c). However, if we take into account its high time complexity demonstrated in Figure 6.4, *LogMap_NoCS* is inferior to the other systems.

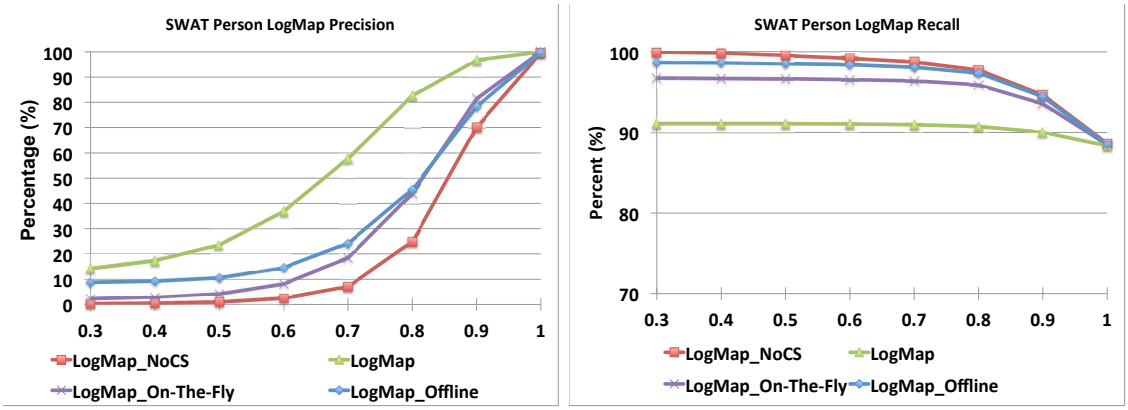
Comparing *LogMap_On-The-Fly* and *LogMap*, *LogMap_On-The-Fly* has clear advantage on recall but is also less precise than the other system, particularly at low thresholds (0.3 to 0.7). Although *LogMap* has higher F1-scores than *LogMap_On-The-Fly* from thresholds 0.3

to 0.8 due to its high precision, *LogMap_On-The-Fly* was able to achieve its highest F1-score of 89.82% at threshold 0.9, 3.19% higher than that of *LogMap* at the same threshold, still showing significant improvement. Considering the fact that *LogMap_On-The-Fly* runs at least 25% faster than *LogMap*, it could be a better choice.

Finally, we see that *LogMap_Offline* achieves higher precision than *LogMap_On-The-Fly*, since it performs candidate selection more aggressively and therefore was able to filter out more potential false positives. At the same time, because *LogMap_Offline* is also more capable of maintaining groundtruth matches as shown in Table 6.2, it was able to achieve nearly as good recall as *LogMap_NoCS* did. Overall, the best F1-score 90.43% was achieved by *LogMap_Offline* at threshold 0.9, compared to the highest F1-scores of 88.38%, 86.63% and 88.93% for *LogMap_NoCS*, *LogMap* and *LogMap_On-The-Fly* respectively at the same threshold. Taking into account the much shorter runtime of *LogMap_Offline*, it appears to be the best choice, at least for this dataset.

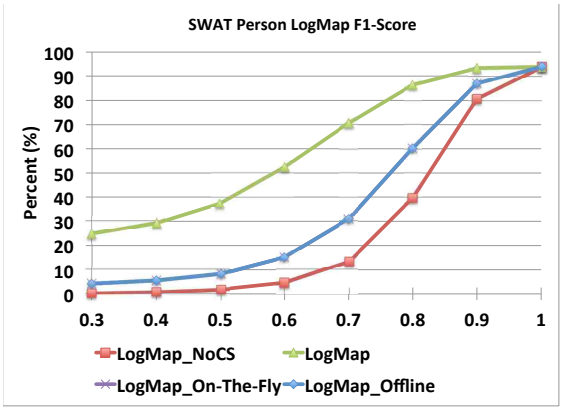
SWAT Person. Figures 6.6(a) to 6.6(c) show the results on the SWAT Person dataset. We observe a very similar pattern to the previous RKB Person dataset for all four systems. Comparing the two our own algorithms, *LogMap_Offline* has higher precision than *LogMap_On-The-Fly* on nearly all thresholds except for threshold 0.9. Similar to what we notice for RKB Person, *LogMap_Offline* was able to maintain a good recall for all thresholds. The best F1-scores for *LogMap_Offline*, *LogMap_NoCS*, *LogMap* and *LogMap_On-The-Fly* are 93.74%, 93.76%, 93.77% and 93.85% respectively, all achieved at threshold 1.0. Although the differences between their best F1-scores are not significant, *LogMap* actually has more improvement in F1-score over the other systems on most of the evaluation thresholds.

RKB Publication. The results are shown in Figures 6.7(a) to 6.7(c) for all four systems. First of all, the *LogMap_Offline* system dominates the other systems on precision



(a) SWAT Person Precision

(b) SWAT Person Recall

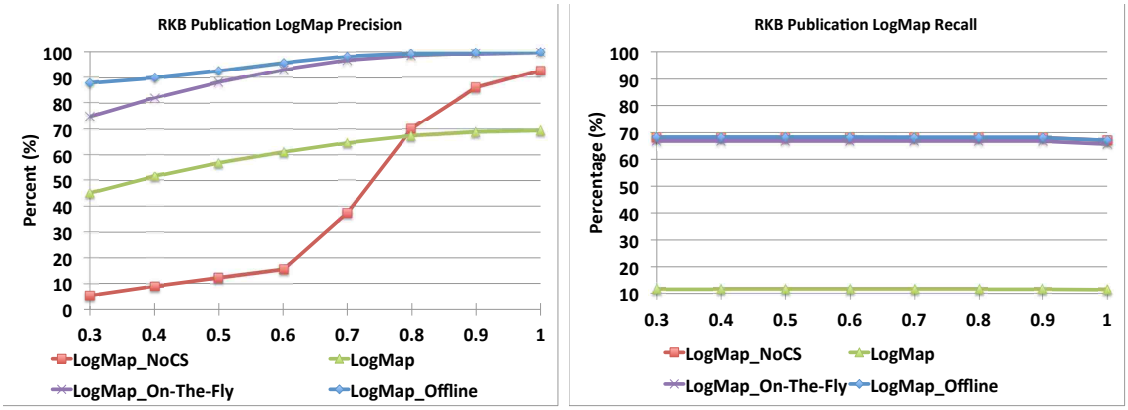


(c) SWAT Person F1-Score

Figure 6.6: Precision, Recall and F1-Score on SWAT Person Dataset by Plugging Candidate Selection into LogMap

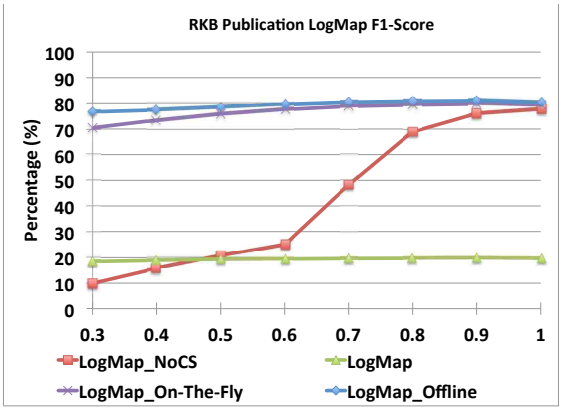
and F1-score on all tested thresholds. This clearly demonstrates the advantage of our offline candidate selection algorithm. Combined with the substantial savings on runtime (Figure 6.4), *LogMap_Offline* is the best choice for RKB Publication.

On the other hand, the results on RKB Publication are different from that of the previous two person datasets, particularly when comparing *LogMap* and *LogMap_On-The-Fly*/*LogMap_Offline*. Here, *LogMap* is dominated by the other two systems on all metrics and tested thresholds. When comparing *LogMap* to its own precision on the two person datasets (Figures 6.5(a) and 6.6(a)), it exhibits a much worse performance. Recall that for *LogMap*, it computes the similarity between all pairs of "label"s between two instances



(a) RKB Publication Precision

(b) RKB Publication Recall



(c) RKB Publication F1-Score

Figure 6.7: Precision, Recall and F1-Score on RKB Publication Dataset by Plugging Candidate Selection into LogMap

and then picks the highest similarity score as the final similarity measure for the two instances. For RKB Publication, "publication-title" was one of the "label" properties; and it is more likely that different publications will have similar titles than different persons will have similar names. This is because many common tokens can occur in titles, especially for papers in related research areas. Combined with the fact that *LogMap*'s candidate selection is not as capable as our on-the-fly and offline algorithms in terms of retaining groundtruth matches, which causes low recall, it was not able to achieve a decent F1-score compared to the other comparison systems.

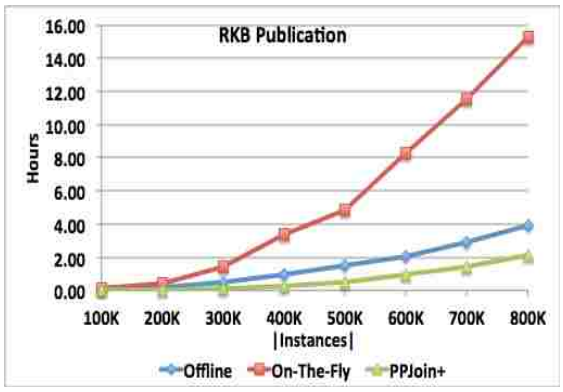
To summarize, we verified our hypotheses that: 1) The candidate selection technique is

an effective step in speeding up the entire entity coreference process; 2) At the same time, candidate selection techniques can also help coreference systems to improve precision by filtering out some of the potential false positives; 3) Finally, by applying appropriate candidate selection techniques, a decent amount of groundtruth pairs can be retained, providing the possibility to achieve satisfying recall and F1-score for the overall process.

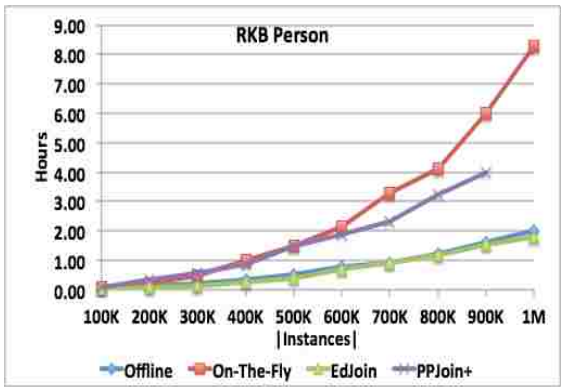
6.4.5 System Scalability

We apply different systems to 100K to 1 million instances to examine their scalability. *All-Pairs*, *PPJoin+* and *FastJoin* achieve very similar results, so we only compare to *PPJoin+*. We also compare to *EdJoin* on both person datasets where it is better on both the number of selected pairs and the overall runtime as shown in Table 5.3; we did not compare to *EdJoin* on RKB Publication since it selects twice as many pairs as *PPJoin+* does on 10K instances and thus is not expected to have better scalability. There are only 500K instances in SWAT Person, and *PPJoin+* does not scale to 1 million instances on RKB Person; also, with 800K instances, *PPJoin+* already shows a clear exponential growth on RKB publication, so we did not test it on the 1 million scale.

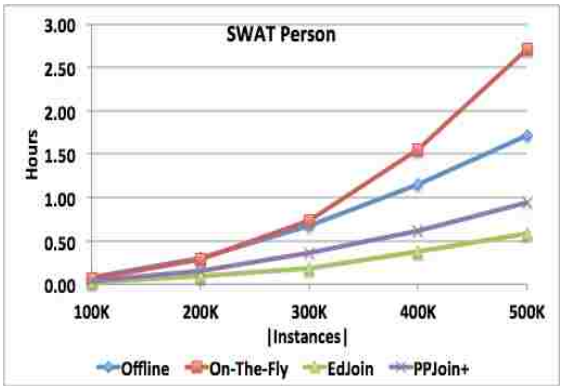
We perform this scalability test for both phases: candidate selection and the entire coreference process. Figures 6.8(a) to 6.8(c) demonstrate the scalability of different systems for the candidate selection phase. We can see that our proposed on-the-fly and offline candidate selection algorithms are not as scalable as the other systems and this is consistent with our previous experiments on small scale testing sets as shown in Table 6.2. However, at the largest compared scale (1M for RKB person, 800K for RKB Publication, and 500K for SWAT Person), the differences between offline and the other state-of-the-art systems are still within an acceptable level: 0.21, 0.83 and 1.12 hours slower than the fastest system



(a) RKB Publication



(b) RKB Person

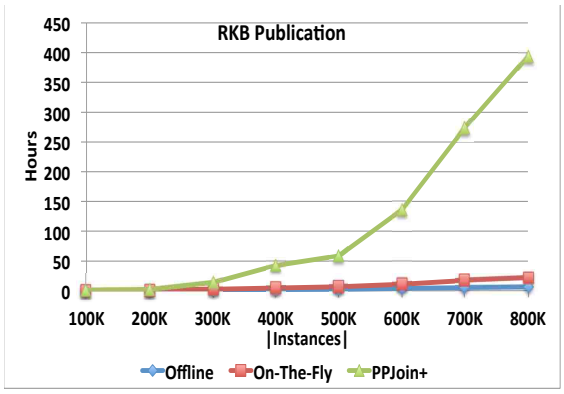


(c) SWAT Person

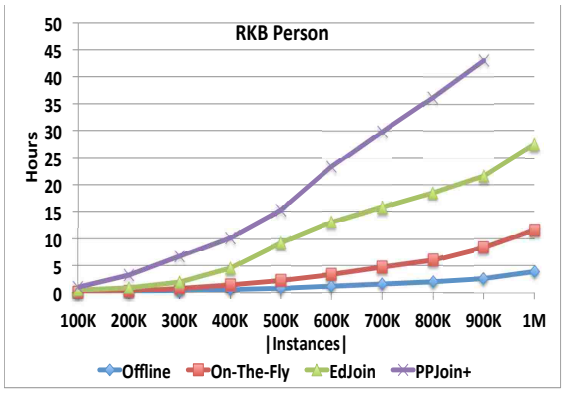
Figure 6.8: Candidate Selection Scalability

on RKB Person, RKB Publication and SWAT Person respectively.

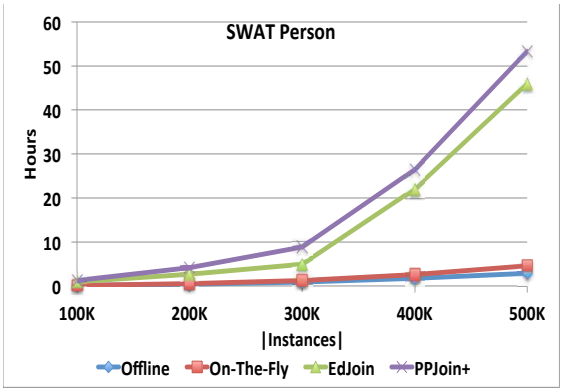
In Figures 6.9(a) to 6.9(c), we show the scalability of all systems for the entire entity coreference process, i.e., candidate selection plus entity coreference. When comparing between our proposed systems, *Offline* shows better scalability than *On-The-Fly*, since it selected much fewer candidate pairs in phase one. At the largest compared scale, on RKB Person, RKB Publication and SWAT Person, *Offline* achieves a reduction ratio of 99.987%, 99.995% and 99.988% respectively; also, when compared to the system that selects the second fewest pairs, *Offline* reduces the size of the candidate sets by a factor of 28 to 36. Compared to *On-The-Fly*, *Offline* is about 3 times faster on RKB Publication and RKB



(a) RKB Publication



(b) RKB Person



(c) SWAT Person

Figure 6.9: Scalability of The Entire Entity Coreference Process

Person, and is about 1.6 times faster on SWAT Person. When compared against the state-of-the-art systems, for the overall process, our proposed systems *Offline* and *On-The-Fly* demonstrate clearly better scalability. Specifically, *Offline* is at about 61, 16 and 18 times faster than those systems on RKB Publication, RKB Person and SWAT Person respectively on the highest compared scale. The other algorithms demonstrate a clear exponential curve; although our two systems are also exponential, it has a much smaller exponent. Considering applying these systems to even larger datasets, the runtime difference could become even more substantial.

6.4.6 Parameter Tuning

In this section, we tune the parameters of our offline candidate selection algorithm to study their impact.

Testing the Similarity Threshold for Choosing a Candidate Pair. In our system, the parameter θ, as part of the refining step, determines whether we should pick a pair of instances to be a candidate. Table 6.4 shows the results by varying θ from 0.5 to 0.8. As

Table 6.4: Testing the Similarity Threshold for Choosing a Candidate Pair. We bold the best scores of each metric among all tested θ values. $|Pairs|$: the number of selected candidate pairs; PC: Pairwise Completeness; RR: Reduction Ratio; $CS\ Time$: time for the candidate selection phase; $Precision$, $Recall$ and $F1\text{-}score$ are for the actual coreference results; $Total\ Time$ is the runtime for the entire process (both phases).

Dataset	Phase	Metric	θ			
			0.5	0.6	0.7	0.8
RKB Person	Candidate Selection	$\|Pairs\|$	46,840	30,560	18,683	**13,790**
		PC (%)	**99.56**	99.54	99.49	99.39
		RR (%)	99.91	99.94	99.96	**99.97**
		CS Time	4.24	4.22	4.22	4.26
	Coreference	$Precision$ (%)	95.01	95.21	95.34	**95.41**
		$Recall$ (%)	**91.00**	90.99	90.99	90.99
		$F1\text{-}score$ (%)	92.95	93.05	93.11	**93.14**
		Total Time (s)	7.79	6.81	6.25	**5.80**
RKB Publication	Candidate Selection	$\|Pairs\|$	43,895	24,532	11,240	**6,270**
		PC (%)	**99.99**	**99.99**	**99.99**	**99.99**
		RR (%)	99.91	99.95	99.98	**99.99**
		CS Time	6.08	5.91	6.25	5.91
	Coreference	$Precision$ (%)	99.64	99.66	99.68	**99.72**
		$Recall$ (%)	**99.80**	**99.80**	**99.80**	**99.80**
		$F1\text{-}score$ (%)	99.72	99.73	99.74	**99.76**
		Total Time (s)	17.67	12.78	9.58	**8.03**
SWAT Person	Candidate Selection	$\|Pairs\|$	49,785	28,337	14,113	**8,059**
		PC (%)	**99.37**	99.31	99.07	98.67
		RR (%)	99.90	99.94	99.97	**99.98**
		CS Time	5.32	5.39	5.39	5.50
	Coreference	$Precision$ (%)	99.40	**99.46**	**99.46**	**99.46**
		$Recall$ (%)	**90.75**	**90.75**	**90.75**	**90.75**
		$F1\text{-}score$ (%)	94.88	**94.90**	**94.90**	**94.90**
		Total Time (s)	12.36	9.56	7.53	**6.78**

we adopt higher similarity thresholds, it is clear that our system selects fewer pairs, which

leads to worse coverage on groundtruth pairs but at the same time enables it to reduce more instance pairs. Although better coverage on groundtruth pairs was observed, this did not translate into significant impact on recall for the actual coreference results. We only notice very minor recall improvement for RKB Person but not on the other two datasets. However, we do achieve better precision for higher θ values, since more potential false positives were filtered out by setting more strict thresholds. As for runtime, because we need to process much fewer pairs for higher θ values, clear runtime savings were achieved for the overall process.

High values for θ are suggested when runtime is critical: the entire process could run faster with certain gain in the coreference results, though we might not have the best PC. When recall is emphasized, low thresholds should be adopted to ensure better PC with tolerable runtime.

The Impact of Less Discriminating Properties and Different Stopping Criteria for Candidate Selection Key Discovery. Recall that β (Line 7 of Algorithm 7) is the threshold for filtering low-discriminability predicates and α (Line 13 of Algorithm 7) is the threshold for F_L score, determining when to stop the key discovery process. As we discussed before, during our candidate selection key discovery process, we remove predicates that are not discriminating enough to avoid selecting too many un-coreferent pairs. For RKB Publication, the "title" predicate itself is sufficiently discriminating (having a discriminability of 0.92) and is used by every instance, giving a F_L score of 0.96; thus, the results will not be affected unless β is high enough such that "title" even gets removed. For SWAT Person, *foaf:name* and *CiteSeer:name* are the only datatype properties used by person instances with discriminability higher than 0.9, so varying β on this dataset from 0.1-0.9 will not affect the results; also, since both properties have a F_L score of 0.85 and 0.05

respectively, setting α lower than 0.85 (which will make *CiteSeer:name* the only property for candidate selection) will not select any candidate pairs.

Table 6.5 shows the results on RKB Person by examining various value combinations of α and β. First of all, when setting α to be 0.6, our system only uses the "full-name"

Table 6.5: Parameter Analysis: α and β on RKB Person. $|Pairs|$: candidate set size; RR: Reduction Ratio; PC: Pairwise Completeness; P and R represent the precision and recall of the actual coreference results; CST and $Total$ are the runtime for candidate selection and the entire process respectively.

Low Dis (β)	Stopping Criteria (α)	*Candidate Selection*			*Coref*		*Time* (s)	
		$\|Pairs\|$	$RR(\%)$	$PC(\%)$	$P(\%)$	$R(\%)$	CST	$Total$
0.4	0.7-0.9	13,790	**99.97**	99.39	**95.41**	90.99	4.32	5.87
0.3	0.7-0.9	13,790	**99.97**	99.39	**95.41**	90.99	4.26	5.80
0.2	0.7-0.9	14,670	**99.97**	99.48	95.20	91.00	5.36	7.09
0.1	0.8-0.9	21,843	99.96	**99.52**	94.71	**91.02**	6.42	8.89
	0.7	20,945	99.96	99.43	94.89	91.01	5.70	7.96
0.1-0.4	0.6	**13,687**	**99.97**	99.39	**95.41**	90.99	**3.99**	**5.56**

predicate for candidate selection, which leads to the fewest selected pairs and shortest runtime. Furthermore, the best Pairwise Completeness was achieved when α equals 0.8 and 0.9, and β equals 0.1. When β is low, we still retain some low-discriminability predicates; while when α is high, our system goes through more iterations to combine more properties. At this combination, we achieve the highest coreference recall and the lowest precision at the same time, since we perform the filtering least aggressively. Finally, as we start to increase β from 0.1 to 0.4, we gradually remove more non-discriminating properties and this enables the system to be more selective for choosing candidates. As a consequence, by sacrificing a little on recall, we gain a relatively large improvement on precision. In the meanwhile, since we are applying the actual coreference algorithm to fewer pairs, we also reduce the overall computational cost with greater β values.

Chapter 7

Towards Linking the Entire Semantic Web: A Pilot Study on the Billion Triples Challenge Dataset

In previous chapters, we presented four different algorithms for detecting coreferent ontology instances. Although those algorithms have been shown to be effective on different datasets, there are still several questions left unsolved. On one hand, in our previous algorithms, we manually specified the comparability between predicates, which is then used to determine the comparability between paths of two weighted neighborhood graphs. This was feasible for the previously used datasets because there were only 92 and 51 predicates in the RKB and SWAT datasets respectively. Manually determining predicate comparability on these two datasets only took about 1 to 2 hours. However, when we start to work on more

heterogeneous datasets where there could be tens of thousands of predicates, manually specifying such comparability becomes nearly infeasible. First, given that the datasets could be really heterogeneous with data coming from various domains, humans may not have the necessary domain expertise to decide predicate comparability. Also, even if we have the required domain knowledge, people may not have the time to do this manually. Therefore, in order to be able to work on really heterogeneous datasets, we need an automatic approach that will produce the comparability between predicates without human input.

On the other hand, although we have tested our algorithms on the RKB and SWAT datasets, our previous testing sets (the ontology instances used for evaluation) are primarily about researchers and their publications. Although the two datasets use different predicates to describe instances, they share some common patterns in their data, e.g., "people have publications", "publications have titles", etc. Therefore, we would like to examine the effectiveness of our algorithms on more heterogeneous datasets that have a wider variety of classes.

In order to meet our testing goals described above, we believe that the Billion Triples Challenge dataset [113] is appropriate (previously discussed in Section 2.3.1). In the rest of this chapter, we will firstly introduce a property matching mechanism for automatically producing predicate comparability, and then discuss how we modify and apply our previously developed entity coreference algorithms to this much larger and more heterogeneous dataset to test their performance.

7.1 A Value-based Property Matching Scheme

The first step towards being able to link the BTC dataset would be an automatic property matching scheme that produces the comparability between various predicates. In the

literature, there have been different types of approaches for matching properties. Stoilos et. al. [114] proposed a property matching algorithm that computes the string similarity between the extracted tokens from the URIs of the properties. Although there are many existing string matching algorithms, such as Levenstein [111], Needleman-Wunsch (assigning different weights to different edit operations) [115] and Jaro-Winkler [116], a novel string matching metric that combines Jaro-Winkler and string overlappings was proposed. Furthermore, reasoning-based approaches were also adopted for ontology alignment [117, 118]. Such systems typically consist of two steps. In the beginning, they will compute syntactic similarity between the labels or extracted tokens of the properties and generate initial mappings. After this, a reasoner is usually utilized to check semantic inconsistency based upon subsumptions and disjointness. Not only schema level information was used for ontology matching, instance data can also be helpful for aligning properties from heterogeneous schemas. Instance-level data can give important insight into the contents and meaning of schema elements, particularly when useful schema information is limited. In general, for such approaches [119, 120, 121], values of the properties from two data sources will be examined and property pairs that share similar values are then treated as matches.

Previous algorithms have been shown to be effective, however they also have some limitations. First of all, only computing the string similarity between extracted tokens from property URIs or labels may not be sufficient. For example, two properties "rdfs:label" and "foaf:name" may not share highly similar strings in their URIs or even labels; however, the former is sometimes used for representing person names. Therefore, only by looking at their URIs or labels would let a property matching system miss this pair of properties and finally cause the recall of the final coreference results to be affected. Logic based approaches have the advantage of being able to help to filter some property pairs that share similar

value spaces but are actually logically disjoint. They may also help to complement property mappings generated by using string matching techniques, when the extracted tokens from URIs are not sufficiently similar. However, one potential drawback is that such approaches heavily rely on the correctness of the utilized ontological axioms. Errors in the logical axioms or ontologies could result in incorrect property mappings.

In our work, we developed a value-based property matching mechanism, which is similar to some of the previous approaches [121]. But different from previous algorithms, we propose a different similarity metric to assist filtering out some false positives. In general, given two datatype properties that we want to match, we extract the tokens from the objects (i.e., literals) of triples of such properties. For object properties, we treat each URI as a token, i.e., we do not tokenize URIs. Once we obtain the tokens, we then calculate the similarity of two properties by examining their tokens and treat them as a match if their computed similarity exceeds the given threshold. Our similarity measures are given in Equations 7.1 and 7.2. Here, p_1 and p_2 represent two properties that we want to match; G is an RDF graph where these two properties are being used; $token\_set(p_1, G)$ is a function that extracts the tokens of property p_1 in graph G and forms a token set.

$$Sim(p_1, p_2, G) = \frac{|\{token\_set(p_1, G)\} \cap \{token\_set(p_2, G)\}|}{\min(|\{token\_set(p_1, G)\}|, |\{token\_set(p_2, G)\}|)} \tag{7.1}$$

$$Ratio(p_1, p_2, G) = \frac{\min(|\{token\_set(p_1, G)\}|, |\{token\_set(p_2, G)\}|)}{\max(|\{token\_set(p_1, G)\}|, |\{token\_set(p_2, G)\}|)} \tag{7.2}$$

To determine whether two properties match, we use two metrics: *Sim* and *Ratio*. *Sim* calculates the similarity between the token sets of two properties. We may notice that our *Sim* function is similar to the traditional Jaccard similarity measure; but instead of using a union in the denominator, we use the min size of the two token sets. This is to

ensure that subproperties can match with their superproperties. Let's take the following property pair "full name" and "label" as a concrete example. For person instances, the "full name" predicate is typically used to represent people's name information; however, in some datasets/situations, data publishers may choose to use the "label" property for representing names, which is perfectly fine and does not violate anything. In this given example, if we adopt the Jaccard similarity measure, then "full name" may not be aligned with "label", because the union of their token sets could be much larger than the intersection. Without this property pair, the performance of the overall coreference results, particularly recall, could be significantly impacted.

In addition to *Sim*, we also employ an additional filtering metric, called *Ratio*. Given two properties, this metric computes the ratio between the size of their token sets. If the calculated *Ratio* value of two properties is below a threshold, then we do not consider the properties comparable. The intuition here is that although we want to be able to cover as many appropriate property mappings as possible by using the min size in the *Sim* function, we would like to filter properties pairs that happen to share common tokens but differ significantly in the size of their token size. To be more specific, let's see the "title" and "keyword" example. Intuitively, we may not say that these two properties are comparable, since they actually represent different semantics. However, if we only use the *Sim* function defined in Equation 7.1, they may be highly similar, because publication titles actually cover the majority of the possible keywords. If we also adopt the *Ratio* metric, the two properties will be filtered out, since the token set of "title" is actually much larger than that of "keyword". We will study the impact of setting different thresholds for both *Sim* and *Ratio* in our evaluation.

7.2 A Modified Graph Matching Algorithm for Detecting Coreferent Ontology Instances for BTC

In order to apply our previous algorithms to the BTC dataset, we made a few modifications through empirical study. First of all, in our previous algorithms and evaluations, we used depth-2 neighborhood, i.e., we stop at depth 2 when getting the neighborhood graph (the expansion process discussed in Section 3.1). However, for the BTC dataset, we only use depth-1 neighborhood graph. For the previous RKB and SWAT datasets, we did not get sufficient literal values at depth 1 to compare to achieve decent recall. However, for the BTC dataset, sufficient literals can be obtained at depth 1, and expanding to depth 2 will greatly slow down the coreference process without gaining much in return.

Another modification is that when working on the BTC dataset, we do not fully follow the open world assumption. Recall our *EPWNG* algorithm described in Algorithm 2 and Section 3.3.5. If the last node of path m of instance a is a URI but it does not match any last node of comparable paths of instance b, we do not add any weight to *total_weight*. While for the BTC dataset, we do apply penalties to such scenarios. Having a comparable path in the other graph indicates that the two neighborhood graphs share similar characteristics (having comparable properties). When the object values of such comparable properties do not match, this indicates a mismatch between the two instances to some degree. Furthermore, in the BTC dataset, because we have the "rdfs:label" and "rdfs:comment" properties, which do not exist for the majority of the instances in RKB and SWAT, it is more easily for two non-coreferent instances to have a decent similarity score. Therefore, in this BTC dataset, we do not fully follow the open world assumption and discount similarity scores for unmatched URI paths. The modified algorithm is shown in Algorithm 9. Compared to the

original algorithm, the only difference is that at Line 10, if there exists any comparable paths for URI path m, no matter whether their end nodes are identical, we assign *path_weight* to the weight of path m and thus it will be used to update *total_weight* (the denominator in the weighted average at Line 16 and 17).

Algorithm 9 Compare(N_x, N_y), N_x is the context N(G,x) and N_y is N(G,y); returns a float number (the similarity of x and y)

1. $total\_score \leftarrow 0, total\_weight \leftarrow 0$
2. **for all** *paths* $m \in N_x$ **do**
3. **if** $\exists path\ n \in N_y, PathComparable(m, n)$ **then**
4. $path\_score \leftarrow 0, path\_weight \leftarrow 0$
5. **if** $E(m)$ *is literal* **then**
6. $path\_score \leftarrow \max_{n' \in N_y, PathComparable(m,n')} Sim(E(m), E(n'))$
7. /* path n' has the highest score with m */
8. $path\_weight \leftarrow (W_m + W_{n'})/2$
9. **else if** $E(m)$ *is URI* **then**
10. $path\_weight \leftarrow W_m$
11. **if** $\exists path\ n' \in N_b, PathComparable(m, n') \wedge E(m) = E(n')$ **then**
12. $path\_score \leftarrow 1$
13. /* path n' has identical end node with m */
14. $path\_weight \leftarrow (W_m + W_{n'})/2$
15. $total\_score \leftarrow total\_score + path\_score * path\_weight$
16. $total\_weight \leftarrow total\_weight + path\_weight$
17. **return** $\frac{total\_score}{total\_weight}$

A last change is that we added special processing for the "wikiPageRedirects" property [1]. In the BTC dataset, we notice that many instances are not associated with any literal values but a single triple on this "wikiPageRedirects" property. Suppose we have an instance a that only has a "wikiPageRedirects" triple. Through our expansion process, we will obtain literal values from the instance that a gets redirected to; thus, those literal paths will be depth-2 paths. However, for instance b, which is coreferent with a, we may have its literal values at depth-1. In this case, when comparing instances a and b, we will not produce a high similarity score, since one of our path comparability conditions is that comparable paths need to be of same length. In our current approach, we simply skip the "wikiPageRedirects"

[1] http://dbpedia.org/ontology/wikiPageRedirects

property, i.e., moving the following nodes in the expanded chains/paths to an upper level. We do this at the coreference time, so that we can easily switch between the two options to compare their performance. To generalize this to other kinds of redirects, we could produce a small mapping ontology. In this ontology, we first define a high level property, called "redirect". We then specify that all properties that represent redirecting are sub properties of this top level "redirect" property and all such properties will not count towards the length of a path. This small mapping ontology will be part of the inputs to our algorithm to facilitate further processes.

7.3 Evaluation

In this section, we present how we prepared the testing datasets, evaluation metrics, comparison systems, and evaluation results.

7.3.1 Testing Dataset Preparation, Parameter Setting, and Evaluation Metrics

Testing Dataset. The BTC dataset provides the groundtruth about instance coreference relationships. We adopt the union-find algorithm [122, 123] to compute the transitive closure of the complete set of *owl:sameAs* statements in the dataset. This generates 1,638,309 *owl:sameAs* clusters; within each cluster, each instance is coreferent to every other instance. Although we evaluate all systems using the provided groundtruth, we did check its quality on 100 randomly sampled instances. To check the soundness of the provided groundtruth of these 100 instances, we examined 126 instance pairs that were missed by our system when the evaluation threshold was set to 0.3, and found that 3 out of these 126 groundtruth pairs should not be made coreferent. As for completeness, with the same 100 instances,

by adopting an evaluation threshold of 0.9, we examined 29 pairs that were reported to be coreferent by our system but were not included in the groundtruth; among these, 4 pairs should actually have been included. With our quality check, though only applied on a small amount of sampled data, I think the existing groundtruth is of decent quality but may need to be augmented to measure recall better.

In order to prepare a testing dataset of size M, we firstly randomly pick M/2 instances that do not have any coreference relationships; we call such instances non-gt instances[2]. We then randomly choose some number of *owl:sameAs* clusters that have M/2 instances in total; we call them gt instances. This way, we have an equal number of gt and non-gt instances. In the collected context information for each instance, we use depth-1 triples where the instance either takes the subject or the object place. On average, there are about 21 triples associated with each instance in our testing set.

Parameter Setting and Evaluation Metrics. Table 7.1 shows the parameters and the values adopted in our experiments. A full parameter analysis will be presented in Section 7.3.6. As for evaluation metrics, we measure the precision, recall, F1-score and runtime of the entire entity coreference process.

Table 7.1: Parameter Settings

Parameter	Value	Description
Sim (Eq. 7.1)	0.3	Predicate comparability: token set similarity
Ratio (Eq. 7.2)	0.03	Predicate comparability: the ratio between token sets
Depth	1	Neighborhood expansion depth limit
Lucene filter	0.2	Threshold for Lucene-based filtering in *EPWNG_Lucene*

7.3.2 Comparison Systems

Here, we describe the details of our comparison systems as listed in Table 7.2. LogMap [72]

[2]Here, "gt" represents **g**round**t**ruth

Table 7.2: Comparison Systems

System	Description
LogMap [72]	Uses manually specified "label" properties for coreference
SERIMI [124]	Automatically determines discriminating properties.
LogMap_Offline	LogMap with our own candidate selection module
EPWNG_Lucene	Use all literal values as context and adopt Lucene for filtering
EPWNG_Offline	EPWNG (Chap. 3) combined with candidate selection (Chap. 6)

and SERIMI [124] are two state-of-the-art algorithms in the Semantic Web for matching ontology instances. LogMap computes the similarity between the values of manually specified "label" properties and picks the highest such similarity as the final similarity score between two instances. SERIMI adopts a similar approach to LogMap. However, instead of letting users to manually specify the "label" properties, SERIMI will automatically compute a discriminability value (which is similar to our weighting mechanism as described in Chapter 3) for each property and pick the ones with the highest discriminability values. Another difference between LogMap and SERIMI is that LogMap requires data to be present locally while SERIMI runs over a SPARQL endpoint. In other words, SERIMI assumes that the data to be matched is hosted in a distributed environment and there is some way through which the data can be retrieved.

For the *LogMap_Offline* system, we replaced LogMap's candidate selection module with our own technique proposed in Chapter 6. As demonstrated in Section 6.4.4, when applied to the RKB and SWAT datasets, this modified version was able to achieve higher recall and F1-score, and was running faster than the original *LogMap* system. The last comparison system *EPWNG_Lucene* adopts our *EPWNG* algorithm (Algorithm 9) for instance matching and utilizes Lucene for performing a simple candidate selection. It indexes the instances on their literal values (called ***Literal Context***); during the coreference process, for each

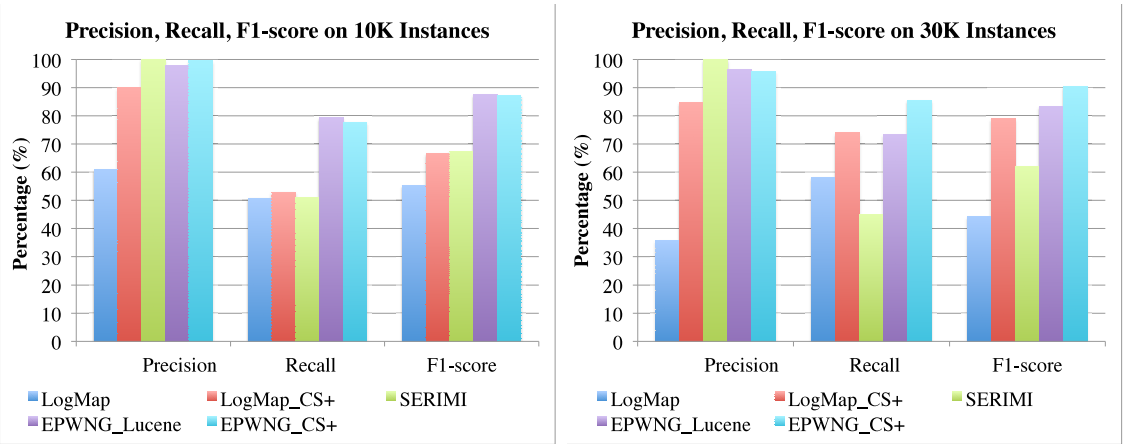
instance i, the system will issue a disjunctive query to the index with all the tokens extracted from i's literal context as terms and retrieve other instances whose Lucene similarity scores to i are above a threshold. Here, the literal context mixes values from different properties, i.e., this simple filtering mechanism does not utilize our property matching results. Finally, the *EPWNG_Offline* system is our proposed one that combines our own instance matching and candidate selection techniques. Both *EPWNG_Lucene* and *EPWNG_Offline* utilize the proposed property mapping mechanism to obtain property comparability; and *EPWNG_Offline* uses the offline candidate selection algorithm presented in Chapter 6.

7.3.3 Evaluating on Small Scale Testing Sets

In our first experiment, we evaluate these systems on some small scale testing sets: 10K, 30K, and 50K instances. Figures 7.1(a) to 7.1(c) demonstrate the precision, recall and F1-score of the 5 comparison systems on each of the three testing sets; Figure 7.2 shows the runtime on the 50K-instance testing set.

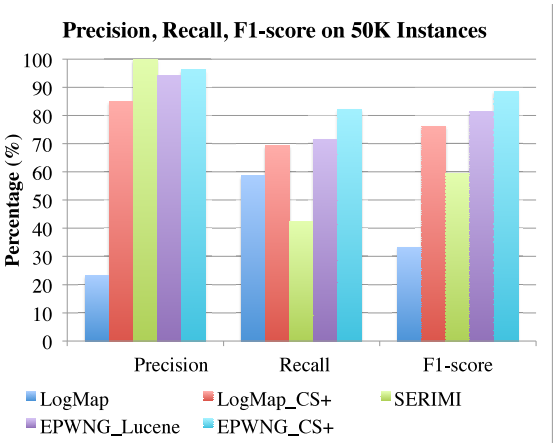
First of all, *LogMap_Offline* dominates *LogMap* on all metrics, which is consistent with our previous evaluation results on the RKB and SWAT datasets (Section 6.4.4). This verifies the effectiveness of our proposed candidate selection algorithm over that of *LogMap* on heterogeneous datasets. Another observation is that *LogMap* gets lower precision as we increase the size of the dataset (particularly from 10K to 30K), because it picks the highest similarity score between values of all specified "label" properties. With more instances, it is more likely to have false positives. Also, although *LogMap_Offline* takes a little longer time than *LogMap* for candidate selection on 50K instances, its much shorter coreference time enabled it to finish the entire process faster.

Furthermore, we compare our own alternatives: *EPWNG_Lucene* and *EPWNG_Offline*.



(a) BTC 10K Results

(b) BTC 30K Results



(c) BTC 50K Results

Figure 7.1: Precision, Recall and F1-score of Comparison Systems When They Achieve the Best F1-score on Small Scale BTC Testing Sets

On one hand, *EPWNG_Offline* only needs about one-third of *EPWNG_Lucene*'s candidate selection time; since it also selects much fewer candidate pairs, *EPWNG_Offline* was able to finish the overall process faster than the other system. Moreover, *EPWNG_Offline* achieved better or comparable F1-score to *EPWNG_Lucene* on all three testing sets. One thing to note is that since our proposed *Offline* candidate select algorithm performs pruning more aggressively, we would expect *EPWNG_Offline* to have better precision and lower recall than *EPWNG_Lucene*. However, for the 30K and 50K testing sets, we actually observe contradictory results. This is because in these figures, we show the precision and

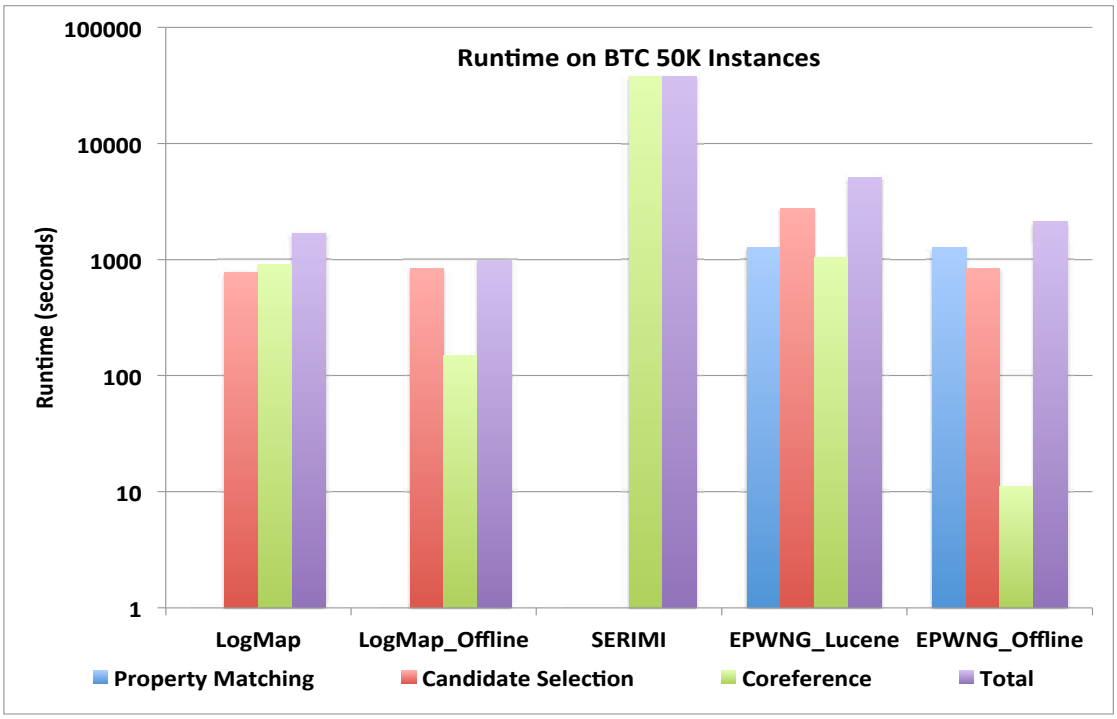
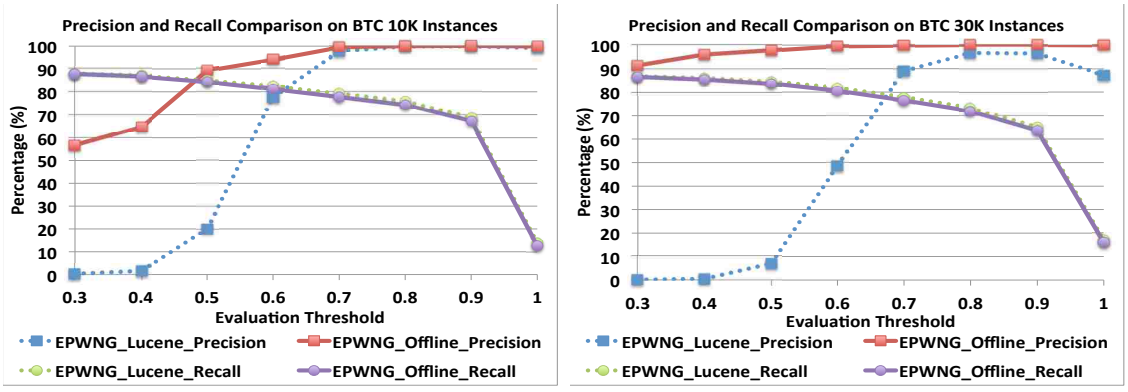


Figure 7.2: BTC 50K Runtime

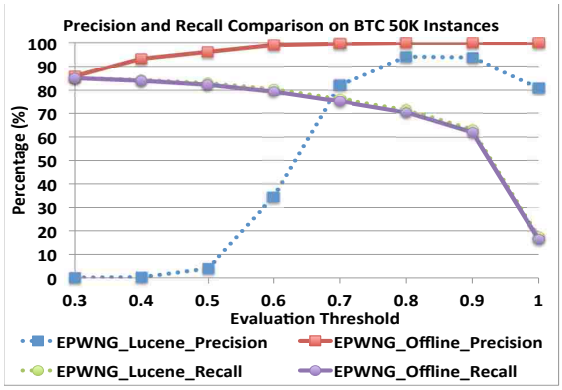
recall where a system achieves its best F1-score; and *EPWNG_Offline* actually has its best F1-scores at a lower threshold than the other. In Figures 7.3(a) to 7.3(c), we show their precision and recall at each tested threshold. And we can see that, at the same threshold, *EPWNG_Offline* generally has better precision and a little worse recall than *EPWNG_Lucene*, which is consistent with our hypothesis.

We also compare *EPWNG_Offline* and *EPWNG_Lucene* to *LogMap* and *LogMap_Offline*. On one hand, both *EPWNG*-based algorithms run slower than the two *LogMap*-based algorithms: 1) The *LogMap*-based algorithms do not need property matching as a pre-processing step, thus saving a significant amount of time; 2) *EPWNG_Lucene* indexes all literal values at depth 1 for candidate selection while *LogMap* only indexes the values of some manually specified properties and therefore runs faster for the candidate selection phase. On the other hand, both *EPWNG*-based algorithms achieve better F1-scores than the other



(a) BTC 10K Results

(b) BTC 30K Results



(c) BTC 50K Results

Figure 7.3: Comparison of Precision and Recall at Corresponding Thresholds on Small Scale BTC Testing Sets

two comparison systems on all three testing sets, although *LogMap_Offline* comes close to *EPWNG_Lucene* on the 30K and 50K testing sets.

Finally, *SERIMI* has the best precision but also the worst recall among all comparison systems on all testing sets. Even being able to achieve the best precision, it may not be the best choice for situations when recall is really critical. In addition, compared to all other systems, it took the longest time to finish the entire process. Considering applying this system on even larger testing sets, it may not scale well.

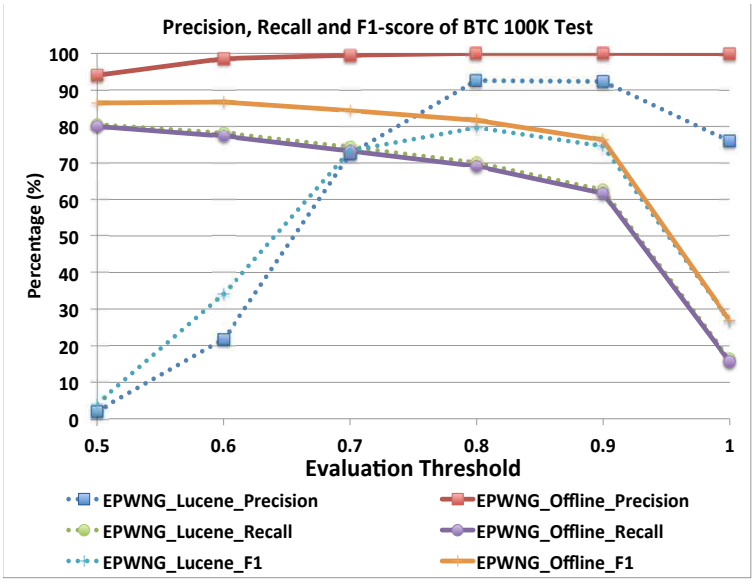


Figure 7.4: Precision, Recall, F1-Score on 100K Testing Sets

7.3.4 Evaluating on Larger BTC Testing Sets

In this experiment, we want to study the effectiveness of the systems on larger testing sets. Following our data selection process described in Section 7.3.1, we produced 10 non-overlapping testing sets, each of which consists of 100K instances. And we will report the standard deviation and test the statistical significance between the results of different comparison systems on these 10 100K-testing sets. Here, we compare the following two systems: *EPWNG_Lucene* and *EPWNG_Offline* that achieved the best F1-scores in previous small scale testings. We demonstrate their precision, recall and F1-score in Figure 7.4 and also compare their runtime in Figure 7.5.

From the results, we can see that *EPWNG_Offline* has clearly better precision than *EPWNG_Lucene*; by being able to achieve comparable recall, it has the best F1-score overall. According to the results on the 10 testing sets, the standard deviation of the F1-scores of *EPWNG_Offline* and *EPWNG_Lucene* on the best evaluation threshold are 2.3958 and 2.6682 respectively. Also, the difference between their F1-scores are statistically significant

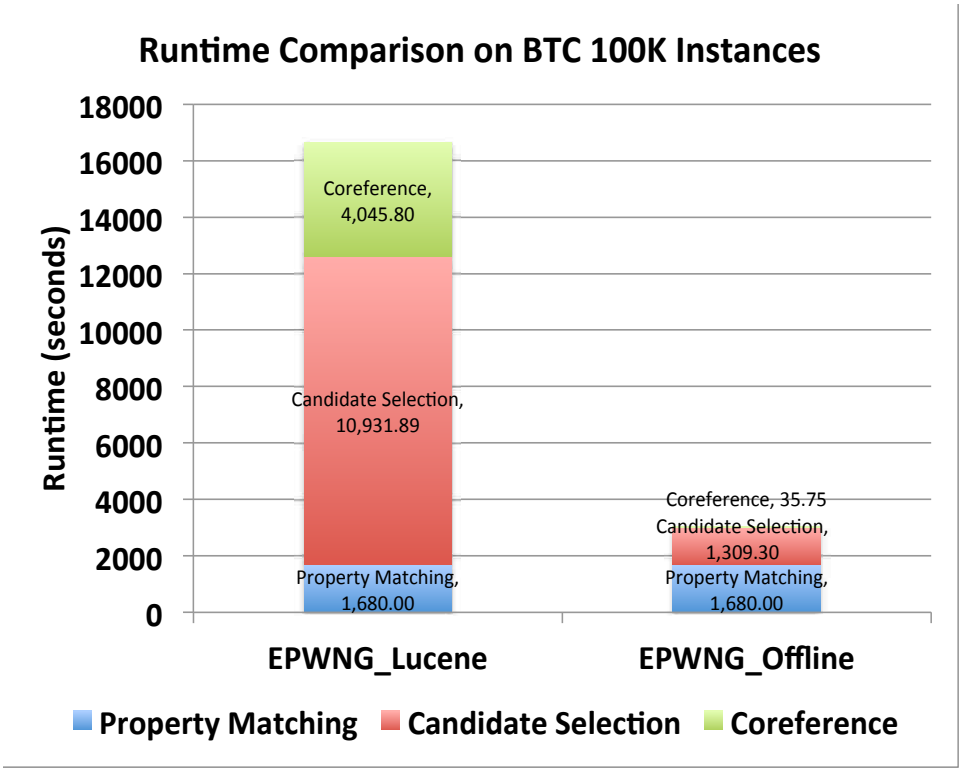


Figure 7.5: Runtime Comparison on 100K Testing Sets

with a P value of 0.0001. As for runtime, both systems require property matching and thus there is no difference on this portion. However, since *EPWNG_Lucene* indexes all literal values and then issues disjunctive queries to find similar instances, it took much longer time for performing candidate selection than querying an index that was built with automatically and carefully selected information. Furthermore, because *EPWNG_Lucene* utilizes more irrelevant information for candidate selection, more candidate pairs were selected, which causes it to also have longer coreference time. On these 100K datasets, *EPWNG_Lucene* is about 5.5 times slower than *EPWNG_Offline*, and this could be translated to a much more remarkable difference for datasets with millions of instances or more (to be presented in Section 7.3.5).

7.3.5 Scalability Test

As we did before on the RKB and SWAT datasets, here, we also perform a scalability test on this BTC dataset but on an even larger scale. In this experiment, we test our proposed systems on up to 2 million instances (randomly selected in the same way as described in Section 7.3.1) to examine how well they scale on large datasets. First of all, in Figure 7.6, we see that by adopting our proposed offline candidate selection technique,

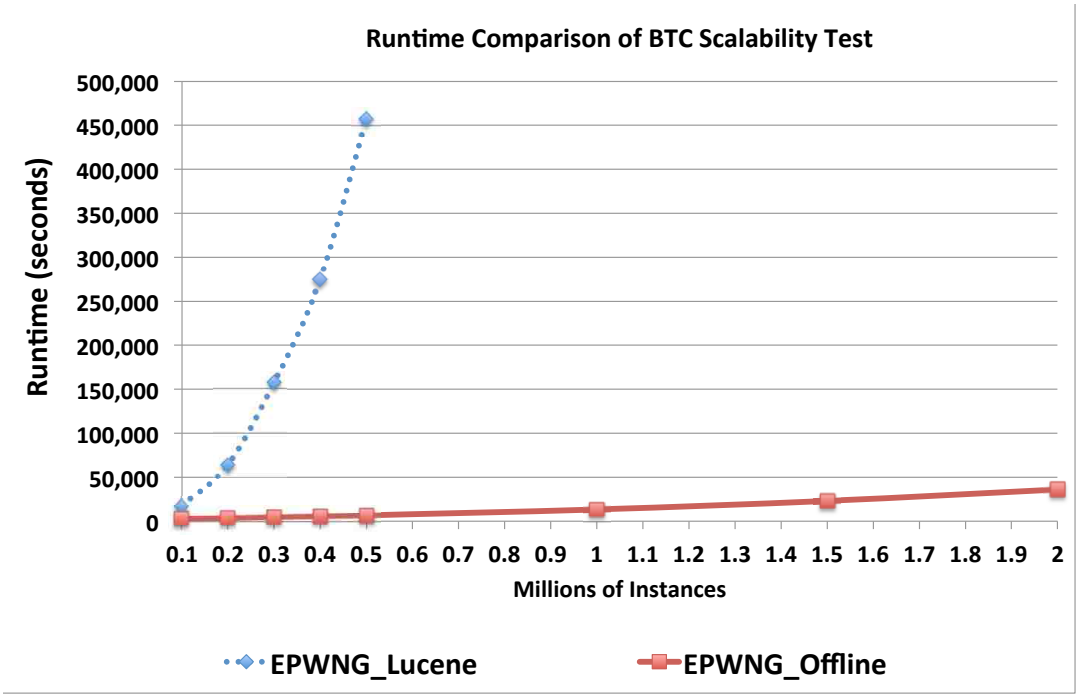


Figure 7.6: Runtime Comparison on 100K Testing Sets

EPWNG_Offline has clear advantage on runtime over *EPWNG_Lucene* which uses simple Lucene-based filtering. For processing 500K instances, *EPWNG_Lucene* already needed 127 hours, therefore was not expected to scale to even larger testing sets. Comparatively, *EPWNG_Offline* only required about 10 hours to finish processing 2 million instances.

Furthermore, as demonstrated in Figure 7.7, *EPWNG_Offline* has significantly better F1-scores on all tested scales. One observation is that as we increase the size of the testing

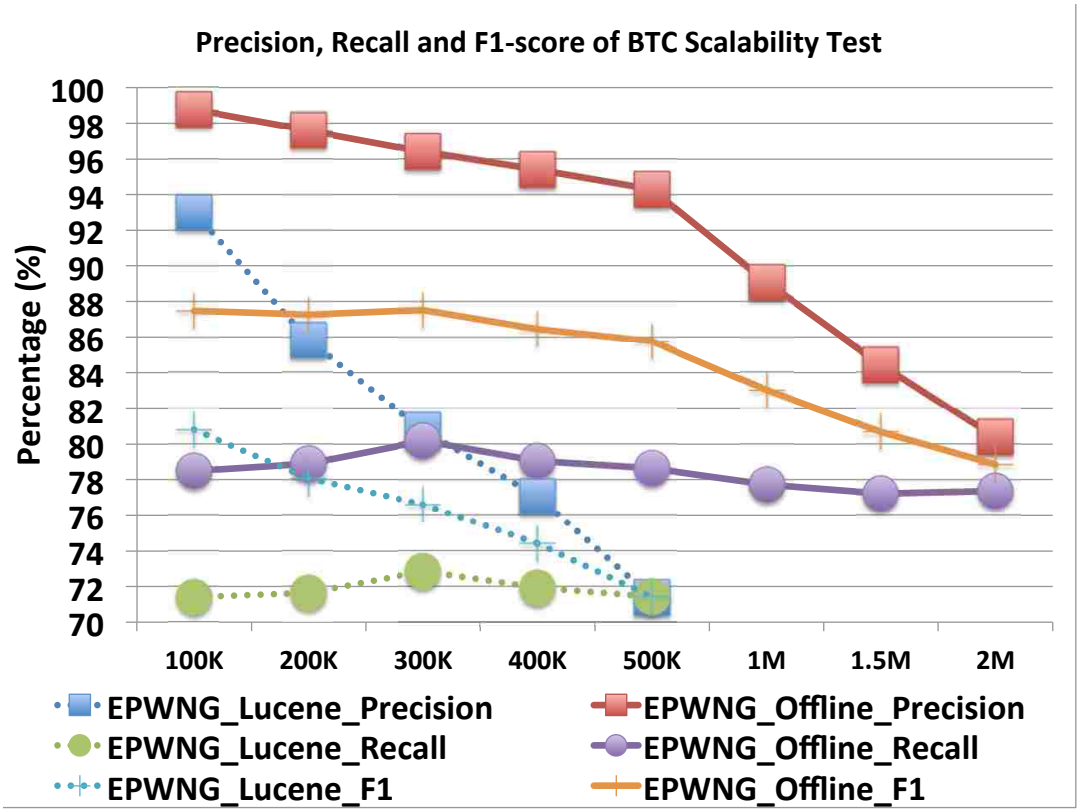


Figure 7.7: Precision, Recall, F1-Score on 100K Testing Sets

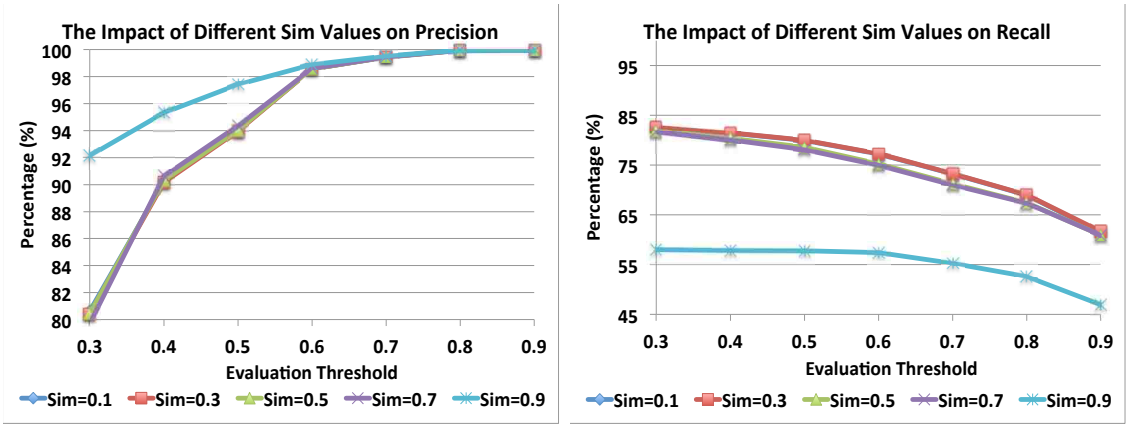
set, both systems start to have lower precision, since it is more likely to have false positives with more instances. However, because our offline candidate selection technique filters uncoreferent instance pairs more aggressively and accurately than the Lucene-based filtering, *EPWNG_Offline* loses its precision more gradually.

7.3.6 Parameter Analysis

In this section, we study the impact of the parameters presented in Table 7.1 and we perform the experiments on the 100K testing sets.

Sim **(Equation 7.1) and** *Ratio* **(Equation 7.2)**. Recall that *Sim* determines whether two properties should be made comparable; and *Ratio* helps us to filter out property pairs

whose token sets are of unbalanced sizes. For the *Sim* parameter, we make two properties comparable if their *Sim* value is higher than a threshold. Figures 7.8(a) and 7.8(b) demonstrate the comparison on precision and recall respectively. In both figures, x-axis rep-

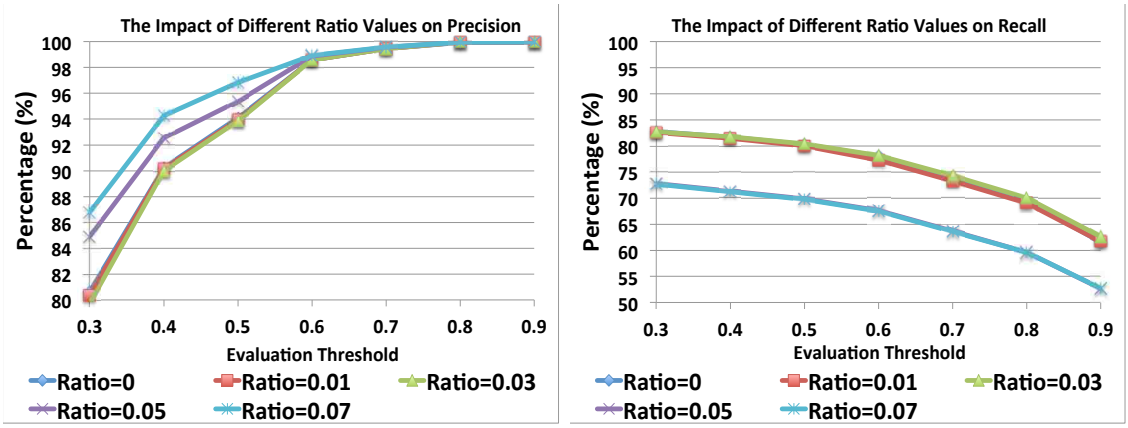


(a) Precision (b) Recall

Figure 7.8: Impact of Applying Different Thresholds for *Sim*

resents different thresholds for evaluating the final coreference results. As shown in Figure 7.8(b), in general, applying higher thresholds for *Sim* will cause the system to miss more true matches, therefore producing lower recall. Unless we set the threshold to be 0.9, i.e., filtering too many truly comparable property pairs, we will still be able to get an acceptable recall. The two curves when *Sim*=0.1 or 0.3 are overlapping. As for precision, we can see from Figure 7.8(a) that adopting different thresholds for *Sim* did not have much impact on precision, except when setting a very high threshold (*Sim*=0.9).

Figures 7.9(a) and 7.9(b) show the precision and recall comparison by setting different thresholds for the *Ratio* parameter. More property pairs will be filtered out by using higher threshold values. On one hand, when we adopt thresholds between 0 and 0.03, no significant difference was observed for precision; while as even higher threshold values were applied, more remarkable improvement was achieved at low evaluation thresholds from 0.3 to 0.5.



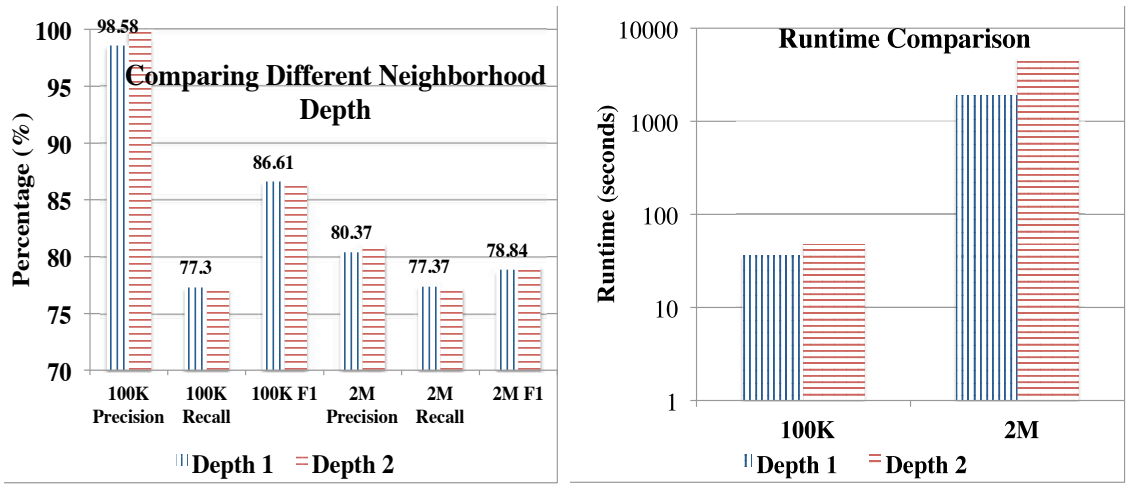
(a) Precision (b) Recall

Figure 7.9: Impact of Applying Different Thresholds for *Ratio*

On the other hand, for recall, unless we apply higher thresholds (i.e., *Ratio*=0.05/0.07), our system was able to detect a decent amount of true matches. It is intuitive that recall continues to drop for higher ratio thresholds, because this filters out matching properties that might have led to a coreference result.

Comparing Different Neighborhood Size. A final test would be to explore the impact of different neighborhood size, i.e., whether we need to collect depth-2 context in our weighted neighborhood graph. We test the two variations, utilizing depth-1 and depth-2 context, on 100K and 2M instances respectively to compare the precision, recall and F1-score. The results here are obtained by employing the *EPWNG_Offline* algorithm.

In Figure 7.10(a), we see that our system achieves slightly higher precision and lower recall when using more context information. By using more context information, there is higher chance that we find more comparable paths and thus the similarity score between two instances is likely to be penalized more. At the same time, with more information, we also have a lower chance to miss a pair of true matches. Overall, there is no significant difference between the F1-scores of depth-1 and depth-2 on the two differently scaled testing



(a) Precision, Recall and F1-score of Coreference Results

(b) Impact of Different Neighborhood Graph Size

Figure 7.10: Examining the Impact of Different Neighborhood Graph Size

sets.

If we look at the runtime comparison in Figure 7.10(b), on 100K testing set, only using depth-1 context sped up the coreference process by 25%. The difference is even more substantial on the 2M testing set, where adopting depth-1 context was 60% faster. Please note that the entire coreference process consists of three steps: property matching, candidate selection, and coreference; and the runtime here is only for the last step. Even if we adopt depth-2 context, considering the overall runtime of 39,029 seconds for 2M instances, the runtime for the actual coreference phase only accounts for about 12%. Depending on the actual application scenarios, it may be beneficial to expand the depth and adopt more context information accordingly.

Chapter 8

Applying Entity Coreference Techniques for Assisting Cervical Cancer Screening and Diagnosis

In previous chapters, we proposed different techniques for detecting coreference relationships between ontology instances. For entity coreference, the algorithms are calculating similarity scores between pairs of identifiers and treat two instances the same if their score exceeds a given threshold. Given such similarity scores, instead of finding equivalent things, we could also potentially use them for finding similarity things (which does not require perfect match). In this chapter, we will describe an interesting piece of work, where we adopt our previous *EPWNG* coreference algorithm for calculating the similarity between patient cases in order to assist the screening and diagnosis of cervical cancer.

8.1 Background and Significance

Cervical cancer afflicts an estimated 12,340 women in the US [1] and 529,800 women worldwide [2] every year. It can be cured if detected during its early stages and treated appropriately. Screening can prevent cervical cancer by detecting Cervical Intraepithelial Neoplasia (CIN), also known as cervical dysplasia. CIN is classified in grades: CIN1 (mild), CIN2 (moderate), and CIN3 (severe). This disease grading is the basis for follow-up treatment and management of the patients. In clinical practice, one of the most important goals of screening is to differentiate CIN1 from CIN2/3 or cancer (denoted as CIN2/3+ in the rest of this chapter), since those lesions in CIN2/3+ will require treatment, whereas mild dysplasia in CIN1 can be observed conservatively because it will typically be cleared by immune response in a year or so.

8.1.1 Clinical Tests for Cervical Cancer Screening

The Pap test is the most widely used cervical cancer screening method[3]. It involves collecting a small sample of cells from the cervix and examining it under a microscope for squamous and glandular intraepithelial lesions (SIL). The result of a Pap test can be either normal or abnormal. Pap tests are effective, but nevertheless require a laboratory infrastructure and trained personnel to evaluate the samples. Furthermore, it is well known in the literature that Pap tests suffer from low sensitivity ($20\% \sim 85\%$) in detecting CIN 2/3+ [125, 126, 127].

The automated Pap test is an alternative to the conventional Pap test (Pap smear).

[1] http://www.cancer.org/research/cancerfactsfigures/cancerfactsfigures/cancer-facts-figures-2013

[2] http://www.cancer.org/research/cancerfactsfigures/globalcancerfactsfigures/global-facts-figures-2nd-ed

[3] http://www.cancer.gov/cancertopics/factsheet/detection/Pap-test

According to the literature, higher sensitivities have been achieved with automation-assisted Pap tests: 79% $\sim$ 82% by using the ThinPrep Imaging System [128] and 81% $\sim$ 86% by adopting the Becton Dickinson FocalPoint GS Imaging System [129]. However, studies regarding the significance of the difference between automated Pap test and conventional Pap test are inconclusive [130, 131].

The HPV test is another screening method that has been used in conjunction with the Pap test either as an additional test or when Pap test results are inconclusive. It has been well established that cervical dysplasia are caused by persistent infection with certain types of human papillomavirus (HPV), thus DNA tests to detect HPV strains associated with cervical cancer (i.e., HPV test) can be used for screening and triage of cervical abnormalities. The sensitivity of the HPV test in detecting CIN 2/3+ lesions varied from 66% to 100% and the specificity varied from 61% to 96% [125]. However, the HPV test is not recommended as a primary screening method, because of its relatively high false positive rate, particularly in younger women [132].

An abnormal Pap test result may also lead to a recommendation for Colposcopy of the cervix, during which a doctor examines the cervix in detail through a magnifying device. If an area of abnormal tissue is seen, the doctor may decide to remove a small sample of tissue from that area (i.e., biopsy) and send it to a lab to be examined under a microscope. CIN can be diagnosed by biopsy. Being a diagnostic procedure and often accompanied by biopsy, Colposcopy is more costly than screening methods such as Pap and HPV tests.

8.1.2 Cervical Cancer Screening with Image Processing

Digital Cervicography [133] is another visual examination method; it takes a photograph of the cervix (called a cervigram) after applying 5% acetic acid to the cervix epithelium.

In the literature, Cervicography has been shown to effectively increase the sensitivity of Pap test in detecting invasive cancer [134] and high-grade (CIN2-3) lesions in patients with previous atypical squamous cells of undetermined significance (ASCUS) or low-grade squamous intraepithelial lesion (LSIL) Pap result [135]. But questions remain regarding its overall effectiveness because studies find poor correlation between visual lesion recognition and disease as well as disagreement among experts for grading visual findings [136].

8.1.3 Overview of Our Approach

The combination of screening and diagnostic procedures has led to the sharp decline of cervical cancer death rates in Western countries. However, in areas that lack laboratories and trained personnel for conducting screening, diagnostic, and follow-up tests, cervical cancer is still one of the leading causes of death in middle-aged women. In 2008, an estimated 275,100 women died from cervical cancer, and nearly 90% of the deaths occurred in developing parts of the world[4]. Consequently, there is a need for less expensive and more automated screening methods, especially those applicable in low-resource regions.

Encouraged by recent developments in computer-assisted diagnosis such as automated Pap tests, in this work, we develop a comprehensive algorithmic framework based on Multi-Modal Entity Coreference for combining various types of information for detecting high-grade (CIN2/3+) cervical lesions. The framework enables the efficient evaluation of the performance of various combinatory tests involving basic patient information, clinical test results and computer assisted cervigram interpretation. With our approach, using Pap test alone gives sensitivity 37% and specificity 96%, and using HPV test alone gives sensitivity 57% and specificity 93%. Furthermore, A novel combinatory test that integrates Pap, HPV,

[4]http://www.cancer.org/research/cancerfactsfigures/globalcancerfactsfigures/global-facts-figures-2nd-ed

pH, patient age, and computer interpretation of cervical images yields 77% sensitivity and 91.2% specificity, a statistically significant improvement over its ablations; this indicates the potential value of utilizing computerized approaches as an adjunctive screening and diagnosis method for pre-cancerous lesions and invasive cancer.

8.2 Data and Materials

We carry out our study on 60,000 digitized uterine cervix images (cervigrams) collected by the National Cancer Institute (NCI) in a longitudinal multi-year study in Guanacaste. NLM MDT (U.S. National Library of Medicine Multimedia Database Tool) [137] is a database tool for accessing these digital cervix images as well as clinical, cytologic, and molecular information at multiple examinations of 10,000 women to study the evolution of lesions related to cervical cancer. The women can be categorized as follows: patients with invasive cancer, patients without cervical lesion at enrollment but later developed disease at follow-up, and healthy women who never developed any pathological changes in the cervix. Some statistics about the dataset are shown in Table 8.1.

Table 8.1: Dataset Statistics

Dataset	Category	Number of All Patients	Used for Experiments
Guanacaste	<CIN2	7669	140
	CIN2	62	60
	CIN3	70	70
	Cancer	10	10

Since our goal is to study the potential of using different types of information for screening, we use data only from the Pap test (i.e., cytologic data), the HPV test, and images (i.e., cervigrams) of resolution 2891 by 1973 pixels. We also consider patient age and pH value in some experiments. The "gold standard" groundtruth against which we evaluate

our disease classification method is histologic data obtained from microscopic evaluation of tissue samples taken during biopsy.

We evaluate our proposed multi-modal patient classifier on 280 randomly selected participants from this Guanacaste database. We had to choose an unbalanced number of patient cases for the four categories because only 10 cancer cases are available in the entire Guanacaste dataset. However, since we are performing a binary classification, i.e., classifying patient cases into one of the following two categories: <CIN2 and CIN2/3+, we do have an equal number of patient cases in these two classes: 140 cases in <CIN2, and 140 cases in CIN2/3+.

8.3 Methodologies

8.3.1 System Overview

Figure 8.1 demonstrates the architecture of our cervical dysplasia classification system:

1. Data Converter. The raw data is stored in a relational database and the data needs to be converted and represented into a hierarchical format needed by our algorithm.

2. Similarity Calculator. Clinical cases are composed of multiple kinds of data including not only the cervigrams but non-imaging data such as other test results and patient history. Therefore, it is important to combine these multi-modal data sources for reliable patient classification. In our system, the calculation of patient case similarity has two sub-components: the data-level similarity involving numeric and symbolic data such as Pap and HPV test results, and image similarity involving cervigrams. We calculate a final similarity between two patient cases by taking the linear combination of the two component similarities.

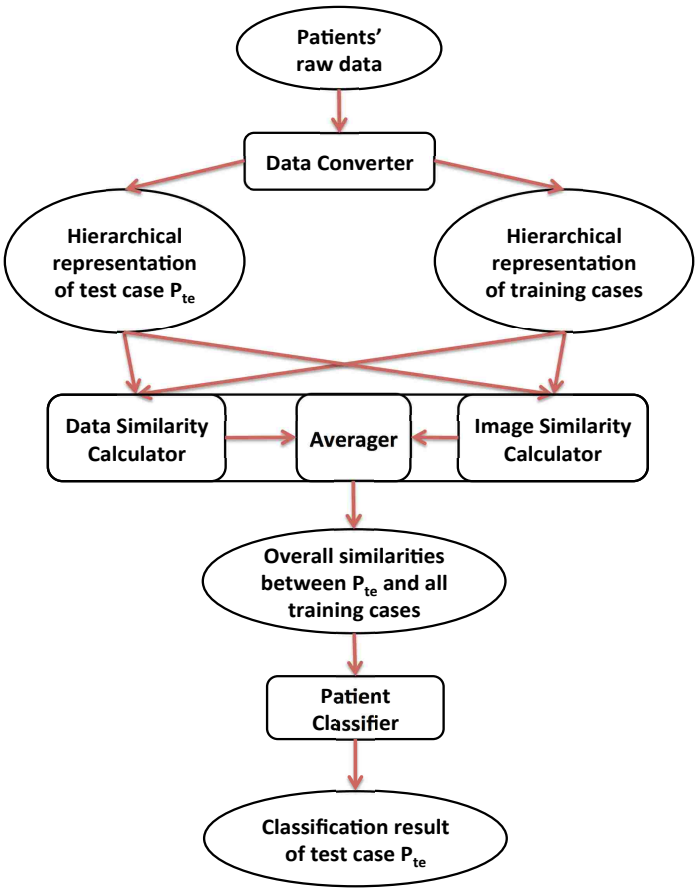


Figure 8.1: Overall System Architecture

3. Patient Classifier. This is the multi-modal aggregation scheme that translates patient case similarity to patient disease classification. We explored different aggregation methods, such as nearest neighbor (i.e., maximum aggregation) and majority voting. And we finally adopt an approach that retrieves the most similar cases from a training database and let the top-cluster training cases to vote to determine the disease grade of a test patient case.

The data structures and algorithms that we will propose are motivated by the challenges presented in the data records. In the database, each patient may have data from multiple clinic visits, and the number of visits differs from patient to patient. Although there are cytology, HPV results and images of almost all patients, the types of cytology and HPV

tests may differ, and there may be missing data for some visits. Thus, the classification system should be able to handle highly unbalanced data, and measure similarity between patients regardless of differences in number of visits and available information.

8.3.2 Patient Similarity Calculation with Entity Coreference Algorithm

Computing Data Level Similarity

In this section, we introduce how we convert the raw data from relational database records to a hierarchical format and we then formally present our approach for measuring data-level patient similarity.

Our data conversion is done in an intuitive way. As shown in Figure 8.2, Patient P_1 has several visits denoted by $VL = \{V_1, V_2, ..., V_i\}$. Take the first visit V_1 of patient P_1 as an example. V_1 stores some basic information about the patient such as age at this visit, and it is also associated with information from some simple tests such as pH value, cervigrams (Cerv), and colposcopy impression. Furthermore, V_1 has two complex clinical tests, C (Cytology) and H (HPV), which are further expanded to have some other simple test results (e.g., $HPV16$ and $HPV18$ are child nodes of H). We use simple and complex to refer to the type of the test result (whether it has sub-test results or not), rather than the complexity of the procedure of the test itself. Finally, CL denotes a node that expands to a list of cervical images (i.e., cervigrams), C_1, ..., C_j, that were taken of the patient P_1 during visit V_1.

We adopt our previous *EPWNG* entity coreference algorithm (shown in Algorithm 10) [16] to compute the similarity of two patients by taking into account their clinical test results. In Algorithm 10, G is a function, retrieving the set of chains for a given patient; *comparable* checks the comparability of two chains; the method l is used to get the leaf node

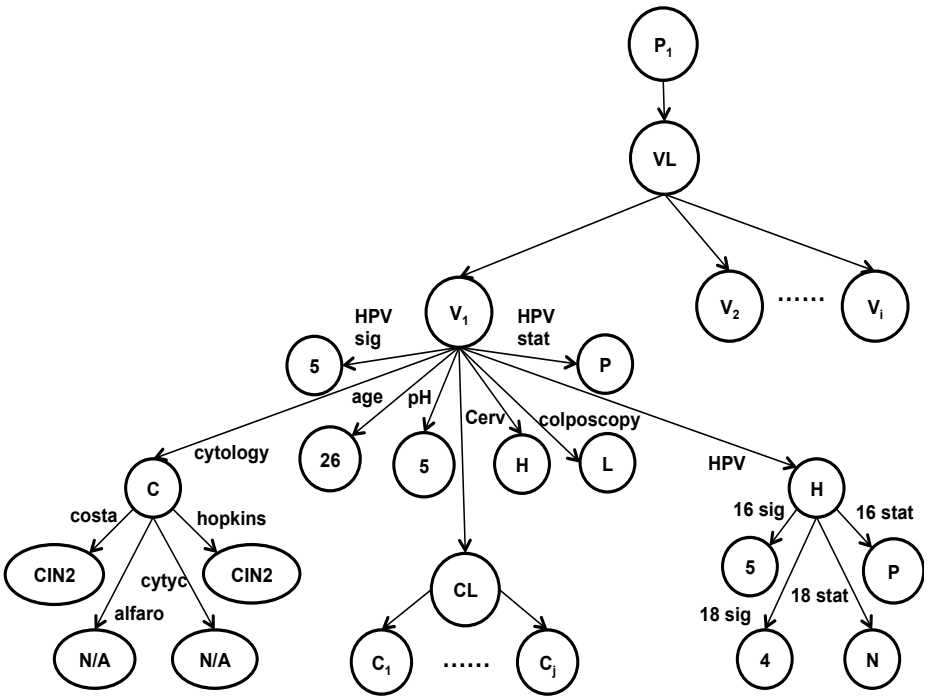


Figure 8.2: Transformed Hierarchical Representation of Patient Data

of a chain; and *Sim* computes the similarity between two leaf nodes. In the hierarchical representation of a patients data (Figure 8.2), a chain is the path from the root to a leaf node. Two chains are comparable if they represent the same type of clinical test determined by the type of edges in the hierarchy. For example, we can have a chain of *patientA* with the following sequence of edges: *Chain_1=(has_visit_list→has_visit→has_hpv→has_hpv16)*. A comparable chain from *patientB* will need to have the same sequence of edges in the chain.

When computing the similarity between the leaf nodes of two comparable chains (the *Sim* function at Line 6 in Algorithm 10), the comparison can be either between numeric values (e.g., patient age and pH value), or strings (e.g., cytology result and HPV status). The similarity between two numeric values is computed with Equation 8.1:

Algorithm 10 $Data\_Sim(G(a), G(b))$, a and b are two patients

1. $score \leftarrow 0$;
2. $count \leftarrow 0$;
3. **for** $c \in G(a)$ **do**
4. **if** $\exists c' \in G(b)$, $comparable(c, c')$ **then**
5. $//t$ refers to the clinical test represented by c
6. $chain\_score =$ $Average_{c' \in G(b), comparable(c,c')} Sim(t, l(c), l(c'))$;
7. $score \leftarrow score + chain\_score$;
8. $count{+}{+}$;
9. **if** $count > 0$ **then**
10. $score \leftarrow \frac{score}{count}$;
11. **return** $score$

$$Sim_{numeric}(n_1, n_2) = 1 - \frac{|n_1 - n_2|}{\max(n_1, n_2)} \tag{8.1}$$

where n_1 and n_2 are two numeric values and the function max returns the maximum between them. The numeric values that we currently handle are all positive numbers.

As for strings, in the cervical cancer domain or the more general clinical care domain, syntactically different strings could be semantically similar or vice versa. For example, "CIN 1" and "CIN 3" are similar in their syntactic representation whereas clinically they are two very different disease stages of cervical dysplasia. Therefore, instead of adopting traditional string matching algorithms (e.g., Jaccard and Edit distance) that coreference algorithms typically use [16, 138, 36, 13], we utilize domain knowledge about the semantic similarity between the result strings of a test. Fortunately, such knowledge is available in the NCI/NLM database. In NLM MDT [137], the possible results of a given clinical test are indexed with some integer numbers indicating their degree or grade. For instance, for Cytology result, "Normal" is given index 1; "ASCUS" is given index 3; "CIN1" is given index 5; "CIN2" and "CIN3" are indexed with 6 and 7 respectively. Thus, utilizing such available domain knowledge, string similarity is computed with Equation 8.2:

$$Sim_{string}(t, s_1, s_2) = 1 - \frac{|Index(t, s_1) - Index(t, s_2)|}{maxDist(t)} \tag{8.2}$$

where t represents a particular clinical test; s_1 and s_2 are two possible results of this test in string format; $Index(t, s)$ retrieves the assigned integer index for string s of clinical test t; $maxDist(t)$ gives the maximum distance between all possible results of test t, which is computed by subtracting the smallest assigned integer index from the largest one.

The similarity measures defined in Equations 8.1 and 8.2 provide a more semantic notion of closeness than simply checking if two numeric values or two strings are identical; and thus they allow us to calculate more accurately the similarity between patients whose clinical test results are similar but not identical.

Computing Cervigram Similarity

Since image similarity computation was not implemented by this author[5], we will not provide all the details. But in general, image features that highlight important visual characteristics in cervigrams were adopted, such as color and texture information.

8.3.3 Patient Classification by Aggregating Image and Data Similarity

Here, we describe how to augment patient data that are traditionally used in clinical testing with cervigram data. Our hypothesis is that the aggregation of these two sources of data should significantly improve the sensitivity and overall accuracy of the classifier in detecting high-grade cervical lesions compared to using either type of data alone.

[5]The author here developed the overall classification framework, applied the coreference algorithm for computing patient similarity and conducted the experiments. Dr. Ed Kim developed the algorithm for image similarity computation [139], which was then utilized by the classification algorithm.

For combining these two heterogeneous types of data, we define an aggregated similarity metric over the data similarity $Data\_Sim(G(a), G(b))$ (computed by Algorithm 10) and the image-based similarity $Image\_Sim(a, b)$ (described in Section 8.3.2). The aggregated similarity metric $sim(a, b)$ for patients a and b is defined in Equation 8.3:

$$sim(a, b) = \alpha \times Data\_Sim(G(a), G(b)) + (1 - \alpha) \times Image\_Sim(a, b) \tag{8.3}$$

where α is a weighting factor. In our experiments (see Section 8.4.1), we set the α value to be 0.5, which means we set equal weights to the data and the image similarities.

Our classification task is a binary classification task: whether a new patient p_n will be classified as $<CIN2$ (Negative) or $CIN2/3+$ (Positive). Conceptually our patient repository can be seen as a case base (CB) of cases, where each case has the form (p, c) where c is the class (i.e., Negative or Positive) of the patient p. We combine lazy and eager learning

Algorithm 11 Classification of a new patient p_n given a case base CB

1. $CB_n \leftarrow \emptyset$
2. **for each** $(p, c) \in CB$ **do**
3. $\quad CB_n \leftarrow CB_n \cup (sim(p, p_n), c)$
4. **end for**
5. $CL \leftarrow KMeansCluster(CB_n)$
6. $tC \leftarrow topCluster(CL)$
7. **return** majorityVote(tC)

approaches [140] for our classifier as shown in Algorithm 11. First we initialize an auxiliary case base CB_n (Step 1). Then, CB_n is filled with pairs, in the format of (sim, c), of similarities between each patient p in the case base CB and the new patient p_n as well as the class label for p (Steps 2-4). We then apply K-means clustering on CB_n; the clusters are

grouped by the similarity scores (Step 5). We then return the class that occurs the most amongst cases in the top cluster (Steps 6 and 7). If there is a tie, a random selection is done among the classes that most frequently occur in the top cluster.

8.4 Experiment

8.4.1 Evaluation Metrics

As stated above, we evaluate our proposed system in a binary classification scenario, i.e., we classify a patient to be either <CIN2 (Negative) or CIN2/3+ (Positive). We measure the accuracy, sensitivity and specificity of our proposed multi-modal patient classifier (see Algorithm 11). The definitions for these metrics are given as follows:

$$Accuracy = \frac{|correctly\ classified\ patient\ cases|}{|test\ cases|} \tag{8.4}$$

$$Sensitivity = \frac{|true\ positive|}{|true\ positive| + |false\ negative|} \tag{8.5}$$

$$Specificity = \frac{|true\ negative|}{|true\ negative| + |false\ positive|} \tag{8.6}$$

where *true positive* refers to the set of patients of the class "Positive" and are correctly classified; *false negative* refers to the set of patients who fall into the class "Positive" but are misclassified as "Negative"; *true negative* and *false positive* are similarly defined.

Following a standard of evaluating machine learning systems, we perform a ten-round ten-fold cross validation on our dataset of 280 patient cases (Table 8.1). In each round, we randomly divide the patient cases into ten folds; in a rotational manner, we use one fold for testing and the 9 remaining folds for training; the testing result for the round is the average of the testing results for each of the ten folds. The final testing result is the average

accuracy/sensitivity/specificity of the ten rounds.

We also test the statistical significance between the results of our proposed system and other systems on our dataset. In this work, we compare each pair of systems through the ten rounds and perform a two-tailed t-test on the two sets of results from the systems.

8.4.2 Multi-modal Entity Classifier vs. Data/Image-only

In this experiment, our goal is to examine the effectiveness of different types of information in the cervical cancer patient classification task, including Cytology, HPV, patient age, pH value and cervigrams (digital images). We first test the individual effectiveness of Cytology, HPV and cervigram, i.e., only using one of the three types of information for classification, and compare their performance. Furthermore, we perform classification by combining different types of information, e.g., using Cytology, HPV, age, pH, cervigram together, and then compare the classification accuracy using these combinatory tests with that of using a single type of information.

In the Guanacaste dataset, the possible values for Cytology include Normal, Rctive, ASCUS, Koil. Atyp, CIN 1/2/3, Micrinv Cncr and Inv Cancer. There are two components to HPV: (1) HPV Signal, which is a floating value ranging from 0.0 to 5.0, and (2) HPV Status, which can be either Negative or Positive. Patient age is a numeric value ranging from 15 to 100. pH value is another numeric value ranging from 1.0 to 14.0.

Please note that the performance numbers in Table 8.2 are the average accuracy, sensitivity, and specificity from the ten-round ten-fold cross validation using 280 patient cases. The overall best accuracy was 84.04% and it was achieved by applying Multi-Modal patient classification using the combination of Cytology, HPV, pH, patient age, and images (C+H+A+P+I). In comparison, using clinical data-only (C+H+A+P), the accuracy was

80.07%; and using image-only (I), the accuracy was 81.93%. The results here are statistically significant with 95% confidence. This demonstrates the effectiveness of combining both data and image similarities for patient classification. For accuracy, a two-tailed t-test on the results between "C+H+A+P+I" and "C+H+A+P" gave a P value of 0.0001 and the results between "C+H+A+P+I" and "I" also gave a P value of 0.0001. For sensitivity, the differences between "C+H+A+P+I" and "C+H+A+P" or "I" are statistically significant with P values of 0.0001 and 0.0003 respectively. Finally, although no difference was observed between "C+H+A+P+I" and "C+H+A+P" on specificity, both of them are better than "I" on specificity, with P values of 0.0026 and 0.0032 respectively. Overall, compared to systems that use textual/numeric data-only or use images-only for patient classification, our proposed system that aggregates the data and image similarities significantly improves accuracy over systems that use fewer information sources

Table 8.2: Performance of Multi-modal (both clinical data and image), Data-Only and Image-Only classifications (C: Cytology; H: HPV; I: Image; A: Age; P: pH)

System	Accuracy (%)	Sensitivity (%)	Specificity (%)
C	66.36	36.79	95.93
H	74.99	56.54	93.43
I	81.93	74.14	89.71
H+I	77.61	63.93	91.29
C+I	72.96	48.07	**97.86**
C+H	76.25	58.64	93.86
C+H+I	80.00	66.43	93.57
C+H+P	78.79	64.29	93.29
C+H+P+I	82.79	72.79	92.79
C+H+A	79.32	65.93	92.71
C+H+A+I	81.79	71.57	92.00
C+H+A+P	80.07	68.93	91.21
C+H+A+P+I	**84.04**	**76.86**	91.21

8.4.3 Effectiveness of Domain Knowledge

In this experiment, we show that adopting domain knowledge (DK) for computing data-level string similarity (Equation 8.2), can significantly improve the results as shown in Table 8.3. For this experiment, we combine Cytology, HPV, pH, and the patient age information together. Since adding domain knowledge will not affect Image-Only classification, it is not compared here.

Table 8.3: Impact of Domain Knowledge on Classification Results
(AC: Accuracy; SE: Sensitivity; SP: Specificity)

System	AC (%)	SE (%)	SP (%)
Multi-Modal	**84.04**	**76.86**	91.21
Multi-Modal no DK	81.82	68.86	**94.79**
Data-Only DK	**80.07**	**68.93**	91.21
Data-Only no DK	76.36	61.21	**91.50**

We can see that adopting domain knowledge helped to achieve significant improvements in accuracy and sensitivity for both "C+H+A+P+I" and "C+H+A+P" classification; the differences here are statistically significant with a P value of 0.0001. This verifies our assumption that in this domain, syntactically different strings could actually be semantically close to each other; therefore, it is important to capture such semantic similarity. In our current work, such semantic similarity is exploited by utilizing the index integers assigned to strings in the NLM MDT database, assuming semantically similar test-result strings will be assigned close indices. In future work, we plan to explore how to compute semantic similarity of two strings by using some dictionaries or ontologies in the domain [141].

8.4.4 Comparing Different Classification Schemes

As presented in Section 8.3.3, our classification scheme involves retrieving similar patient cases from a case database, performing K-means clustering on the similar cases, and adopting the class label as voted by a majority of cases in the top cluster. For K-means clustering (Step 5 of Algorithm 11), we tried different K values and found K=5 to be a good choice given our training case base of size 252. Note that our training case base has a size of 252 because there are 280 cases in total and in each round of 10-fold cross validation, 1 fold (28 cases) is used for testing and 9 folds (252 cases) are used for training.

Alternatively, instead of majority voting by cases in the top cluster, we could compute the average (or maximum) similarity between a test case and all training cases in each class, and then assign to the test case the class label with maximum similarity. We compared these alternatives in Table 8.4 for Multi-Modal classification using Cytology+HPV+pH+Age (as Data) as well as images. The results show that majority voting by top cluster gives both the best accuracy and sensitivity. Statistically, the differences between Cluster and other classification schemes (Avg and Max) on accuracy and sensitivity are significant with a P value of 0.0001.

Table 8.4: Performance Comparison for Multi-modal Classification with Different Classifiers
(AC: Accuracy; SE: Sensitivity; SP: Specificity.)
(Cluster: majority voting by cases in top cluster;
Avg: average similarity to cases in each class;
Max: maximum similarity to cases in each class.)

Classifier	AC (%)	SE (%)	SP (%)
Cluster	**84.04**	**76.86**	91.21
Avg	82.46	73.14	**91.79**
Max	79.29	71.29	87.29

8.5 Discussion

Through our evaluation, we have shown that integrating image interpretation with other clinical information can improve the accuracy in differentiating low-grade cervical lesions from high-grade lesions and invasive cancer. By using only digital cervigram images, our proposed system achieved 74.14% sensitivity for detecting CIN2+ lesions; and by using images and 4 other clinical test results (Cytology, HPV, pH, age), our system achieved 76.86% sensitivity. In comparison, the commonly used Pap test screening highly depends on the expertise of laboratory personnel as well as workplace infrastructure; its sensitivity for detecting CIN2+ lesions varies widely in different geographic regions: 22% in Chile [126], 55% in Canada [127], 57% in Africa and India [142], 63% in Costa Rica [143] , and 77% in the United Kingdom [144]. As one can see, the sensitivity levels of our system match the best results reported in clinical literature, which shows the potential of using our system for cervical cancer screening and diagnosis.

Another interesting observation from our experimental results in Table 8.2 is that adding patient age and pH value information improved the systems sensitivity by sacrificing some specificity and therefore enabled the system to achieve better overall classification accuracy when combining all information together. Comparing "C+H+A+I" to "C+H+I" and "C+H+A+P+I" to "C+H+P+I", one can see that patient age information helped to improve sensitivity and accuracy. In fact, patient age has been an important factor used in cervical cancer screening guidelines for average-risk women [145]. For example, it is recommended that women aged less than 21 should not be screened; for women between 21 and 29 years old, Cytology alone should be used every 3 years without HPV co-test; for women between 30 and 65 years old, Cytology should be used every 3 years with HPV co-test every 5 years; and it is recommended that cervical cancer screening can stop for women aged >65

years with adequate screening history. To seek further explanation for the improvement in classification accuracy by adding patient age as a feature, we compiled statistics about patient age from our 280 randomly selected patient cases, as shown in Table 8.5. Here we can see that the distribution of disease does differ significantly from one age group to another, thus making age a useful feature when comparing patients and performing disease classification.

Table 8.5: Patient age distribution in 280 randomly selected patient cases.)

Category	<21	21-29	30-40	41-65	>65
<CIN2 (Negative)	0	9	48	59	24
CIN2/3+ (Positive)	1	38	53	42	6

We also examined the computational complexity of our multi-modal classification system. For the image part, the similarity between images can be finished in around 3 seconds [6]. And then the multi-modal entity coreference algorithm takes about 33 milliseconds to classify the patient case using all five sources of information on a laptop computer with 4GB memory and 2.0GHz quad-core CPU. Theoretically, the complexity of our multi-modal entity coreference algorithm depends on the number of chains in a patients data tree. In the worst case, suppose a tree has p chains and each chain is comparable with all chains in the other tree, the complexity for comparing two trees is then O(p^2). For classification, each test case is computed against all training cases and suppose we have n training cases in total, then the complexity for comparing a test case with all training cases is O($n * p^2$). Once the similarity scores are computed, it takes $O(nlog^n)$ time to sort the scores and obtain the top cluster of most similar cases for classification.

To further improve the accuracy of our classification process, it would be interesting to explore assigning different weights to different features, i.e., automatically determining the

[6]This is from Dr. Ed Kim's experiments

weights for different types of information: age, HPV results and Cytology results. Also, currently we treat data similarity and image similarity equally important; in future work, we could tune the α weight parameter in Equation 8.3, so that the weights of the two different similarities can be adjusted to optimal levels. Finally, in addition to performing binary classification, more fine-grained disease grading could be employed for better assisting the screening and diagnosis process.

Chapter 9

Conclusion and Future Work

We are on the verge of an explosion of RDF data. Individual users, academic institutions and businesses now publish data in Semantic Web formats. BestBuy.com has started a complete RDF/XML dump of their products and price information to the Web of Linked Data, using the GoodRelations vocabulary for e-commerce. The data dump is updated on a daily basis and contains detailed descriptions for roughly 450,000 individual items. With about 60 triples per item, this totals to about 27 million RDF triples[1]. Global media agencies like the British Broadcasting Corporation (BBC) and the New York Times (NYT) have also began to expose their huge stores of data in RDF and link them to other Semantic Web vocabularies and repositories. BBC Music provides RDF information about musicians and it has about 388,398 artist pages, and 93,912 artist to artist relationships[2]. New York Times publishes data about various types of entities in Semantic Web format[3], currently describing about 5,000 people, 1500 organizations and 2000 locations respectively. Using natural language processing and information extraction techniques, academics have also

[1] http://stores.bestbuy.com/sitemap.xml

[2] One example is here: http://www.bbc.co.uk/music/artists/79239441-bfd5-4981-a70c-55c3f15c1287.rdf

[3] http://data.nytimes.com/

created an RDFized Wikipedia, called DBpedia[4] [47], that describes about 4 million entities with a total of 470 million triples. Freebase[5] [49], similar to DBpedia but with a much larger amount of 1.2 billion triples, covers entities of various types, ranging from entertainment, sports to arts, musics, books and to medicine and biology. It is also promising to see that the search engine giants, Google and Yahoo, have began to incorporate RDFa[6] (a means of embedding RDF directly in HTML pages) in their search results. Of particular interest, the idea of Knowledge Graph by Google[7], which is essentially the idea of the Semantic Web and used to describe things and their relationships with a graph model, has been implemented and adopted to power Google's online search. Other search engine giants, such as Yahoo and Bing, are also building their own knowledge graphs to facilitate search. Although not explicitly Semantic Web, because a knowledge graph is a large graph with heterogeneous links, research outcome in the Semantic Web field can be directly applied.

The original Semantic We vision of widely-distributed machine readable data is becoming a reality, while exposing data as RDF is only an important first step. With RDF data, we could then find the relationships between things, such as finding a list of researchers, seeing the organizations those researchers are affiliated with, and other information as well. However, in order to actually achieve the linked data vision we must create explicit RDF links between data items within different data sources. This provides the means by which we can discover more information about a given entity that is not available within any single knowledge base. For example, after the New York Times index terms had been converted to RDF, developers explained that "even though we can show you every article written about "Colbert, Stephen", our databases can't tell you that he was born on May 13, 1964, or that

[4]http://dbpedia.org/About
[5]https://developers.google.com/freebase/data
[6]http://www.w3.org/TR/xhtml-rdfa-primer/
[7]http://www.google.com/insidesearch/features/search/knowledge.html

he lost the 2008 Grammy for best spoken word album to Al Gore"[8]. In order to enable this kind of sophisticated query, we must develop rich linking hubs to store information about entities from multiple and very possibly heterogeneous data sources, so that queries can actually go across different data sets.

To link instances between different data sources, one intuitive idea would be to find the same identifier across those data sets. However, in reality, different data publishers are indeed using their individual ontologies to represent and publish their data on the web, i.e., Semantic Web data is heterogeneous in the sense that the data is represented with different classes and predicates at the schema level. More than that, at the data level, the same real world entity can actually be represented by syntactically distinct URIs when described in different data sources. This information heterogeneity problem is a major barrier for end users who wish to process and consume the data effectively.

9.1 Summary of Research

In order to facilitate data consumption in the Semantic Web, without compromising the freedom of people to publish their data, one critical problem is to appropriately interlink such heterogeneous data. This interlinking process can also be referred to as Entity Coreference, i.e., finding which identifiers refer to the same real world entity. In this dissertation, a series of algorithms have been proposed that primarily target on how to automatically, precisely, and comprehensively detect equivalent (i.e., *owl:sameAs* links) Semantic Web instances from different data publishers in a scalable manner. In Figure 9.1, we show the

[8]http://open.blogs.nytimes.com/2009/10/29/first-5000-tags-released-to-the-linked-data-cloud/?_r=0

overall architecture of the entire coreference system, so that each of the accomplished algorithms can be placed in context.

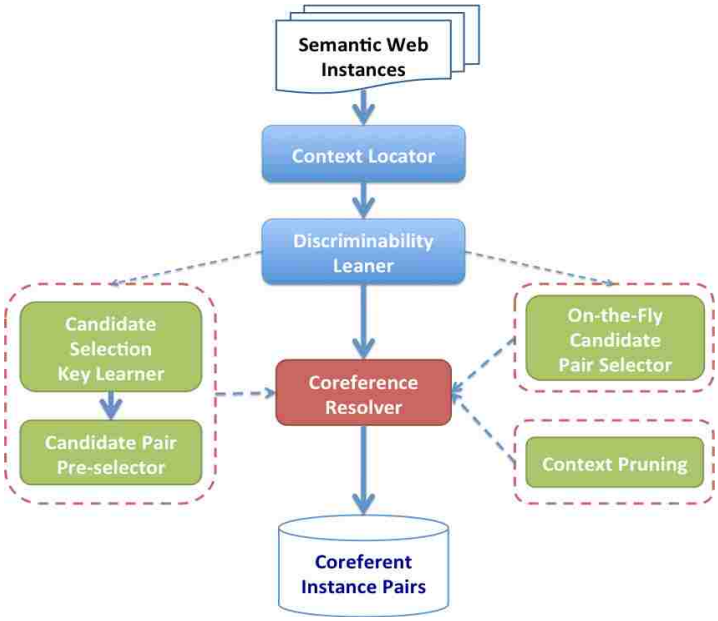


Figure 9.1: An Overview of the Developed Entity Coreference System

As the core of the proposed entity coreference system, a novel entity coreference algorithm is proposed to detect equivalences between ontology instances, which focuses on achieving high precision and recall. For a given instance, we extract its neighborhood graph from the entire RDF graph through an expansion process and we end up having a set of paths starting from this instance and ending on another node in the RDF graph. We then compute the discriminability of each triple, taking into account its predicate, and such discriminability is then discounted according to the triples distance to the root node (the ontology instance), constructing the weighted neighborhood. A bag-of-paths approach is finally adopted to compute the similarity score between a pair of ontology instances. Our system outperforms the state-of-the-art when applied to three benchmark datasets for ontology instance matching with 1% to 4% higher F1-scores. To improve the efficiency of a single pairwise comparison between ontology instances, a sampling and utility function

based pruning technique is proposed. By estimating the potential contribution of RDF nodes in the context of an instance, the system prunes context that would not make significant contribution to the final coreference similarity measure. Applying this pruning technique leads to a speedup factor of 30 to 75 on three RDF datasets.

In order to scale entity coreference systems to the scale of the Semantic Web data today [146], an on-the-fly candidate selection algorithm is presented to filter instance pairs that are not likely to be coreferent by taking advantage instances' matching histories. We discard candidate pairs of instances that are not sufficiently similar to the same pool of other instances. A sigmoid function based thresholding method is proposed to automatically adjust the threshold for such commonality on-the-fly. We verify the effectiveness of our algorithm by comparing to nine state-of-the-art systems and show that our algorithm frequently outperforms those systems with a runtime speedup factor of 18 to 24 while maintaining competitive F1-scores. For datasets of up to 1 million instances, this translates to as much as 370 hours improvement in runtime. One problem of this method is that it was not able to maintain perfect coverage on true matches while performing filtering; although this did not impact the final coreference F1-scores of our proposed system significantly, the recall of our overall system was indeed lower (about 1%) than that of the state-of-the-art on the two RKB datasets (Table 5.3).

In order to improve the capability of covering true matches, a candidate pre-selection algorithm is further proposed. In this algorithm, we select candidate instance pairs by computing a character-level similarity metric on discriminating literal values that are chosen using domain-independent unsupervised learning. Instances are then indexed on the chosen predicates' literal values to enable efficient look-up for similar instances. When evaluated

against state-of-the-art candidate selection algorithms, our proposed algorithm was demonstrated to be more capable of reducing the size of the candidate set while at the same time it was also able to maintain comparable coverage on true matches. By applying an actual entity coreference algorithm to the selected candidate pairs, the overall entity coreference process was sped up by a factor of from 16 to 61 compared to the best-performing state-of-the-art system on up to 1 million instances. Furthermore, by applying our candidate selection technique prior to an entity coreference algorithm, we achieved an improved (0.3% to 0.5% higher) F1-score, due to the fact that candidate selection helped to filter out some of the potential false positives. The results demonstrate the effectiveness of our proposed algorithm in scaling the entire coreference process.

Finally, in order to be able to work on datasets with a large number of predicates, where manual specifying predicate comparability is not feasible, we developed a value-based property matching mechanism. This approach extracts the token sets of two properties and treat the two properties as comparable if there is sufficient overlapping between their token sets. With this approach, we pre-compute the predicate comparability on the BTC 2012 dataset, and apply our proposed *EPWNG* and *Offline* algorithms to testing sets of various scales. When compared against state-of-the-art and comparison coreference systems in the Semantic Web, our algorithm achieved about 7% higher F1-score, and also demonstrated a speedup factor of 66 in a scalability test of up to 2 million instances.

In addition to developing novel coreference algorithms, we also applied the algorithms to an application scenario for cervical cancer patient classification. Instead of finding equivalences, we utilized the computed similarity scores for grouping similar patient cases for cervical cancer screening. Utilizing our *Bag-Of-Chain* approach, by combining different types of information, our system achieved higher classification accuracy than its ablations.

The big theme of the research work here is to evolve the Semantic Web into a real interlinked scenario. I hope the outcome of the research could help us to achieve this goal to facilitate high level applications, such as intelligent and efficient search, and would benefit individuals by improving their ability to utilize the Internet for information needs. In addition, I hope it will also benefit the economy by allowing businesses to better interoperate, both nationally and internationally, with their knowledge globally connected.

9.2 Future Work

Although the current algorithms have been shown to outperform several state-of-the-art algorithms on datasets of various scales and different levels of heterogeneity, I would like to shed light on some potential future directions.

9.2.1 Collective/Multitype Entity Coreference

In our current algorithms, we try to detect the coreference relationships between instances of a single "type", e.g., Person, Publication, etc. Although for the BTC dataset, we mix all types of instances together and compute their coreference relationships, we still treat them as instances of *owl:Thing* (the top class of all other classes). However, given that LOD covers datasets from various domains, one might imagine how would the coreference results of one type of instances impact the others. For example, academic publications and researchers are generally correlated in academic datasets. Suppose we start from publications (since titles are generally very discriminating), could we then be able to achieve higher recall on matching person data by being able to provide better hints for person instance pairs with non-discriminative names (due to abbreviation, misspelling, etc.) but sharing coreferent publication instances (represented with syntactically distinct URIs) in their context? One

step further, could we come up with approaches to automatically prioritize the domains to process, i.e., determining which domains should be processed first so that the other domains could benefit most? For scalability reasons, we could start with the existing linkages in the most influential domain instead of detecting everything from scratch. Since the existing links in LOD are of questionable quality [9], a lightweight verification step might be needed to first check the correctness of such links.

9.2.2 Comparing Non-text and Non-discriminative Property Values

Another problem that I would like to explore is about how to better handle non-text data, such as number and date. In my current algorithms [16, 21], the calculation of instance similarity heavily relies on the utilization of string matching algorithms. I adopted Jaccard, JaroWinklerTFIDF and Edit Distance to compute the similarity between person names, publication titles, etc. However, using such string matching algorithms to compare non-text values may not be the best choice. For example, two dates *09/02/2011* and *09/02/2001* should have a similarity of 0 while a string matching algorithm might give a very high score for them. Another issue about date and number information is that they could use different precision and units (though the units may not be explicitly stated).

Furthermore, in prior work [16, 21, 103, 19], the data we try to integrate generally contains some discriminative labels, e.g., names for people, hotel and restaurant and titles for publications. The question is what if we try to address domains that lack such discriminating labels? Maybe all predicates would then have relatively the same weight and thus *EPWNG* erroneously treats every piece of information the same? Or maybe all datatype properties will be selected for candidate selection and therefore no reduction will be achieved by having to deal with every single triple? One preliminary idea to handling non-discriminative data

is to combine values from multiple properties, expecting the combined values could be more discriminating than that of any individual property.

9.2.3 Handling Data Quality Issues

Data quality issues should be taken into consideration in developing future coreference systems. Although a large amount of heterogeneous data is converted into RDF format by a variety of tools and then made available as Linked Data, e.g., DBpedia, Freebase, RKB, DBLP, Data.gov, etc., not all the data here is of high quality [147]. During the creation or conversion of this data, numerous data quality problems can arise, such as misspellings and errors; and such data quality issues may impact the coreference results. The proposed candidate selection algorithm (Section 2.4) performs exact lookup into the instance index on each query token of a given literal value; therefore, it may miss a certain amount of groundtruth when the object values of the selected datatype properties of coreferent pairs do not share any common tokens, which could be caused by data quality issues. For example, two coreferent person instances may have the following values for the *full_name* property: *J. Smith* and *John Smth*. One of the names uses first initial while the other uses full first name; also, there is one *i* missing in the last name of the second value. When querying the instance index with either of two values, the other one cannot be retrieved since they do not share any common tokens.

To handle such situations, for specific domains, we could design specialized methods. For instance, for person names, in addition to indexing first name, we may also index the initials at the same time. However, to be general, one possible solution would be to adopt fuzzy matching techniques [148]. For the example given above, we employ Edit distance such that instances will be retrieved when the distance between an index term and a query token

is less than a pre-defined threshold. Thus, the token *Smth* could match *Smith*. However, one important problem here is that such fuzzy matching must be performed in an efficient way.

One potential approach is to group similar tokens together by utilizing some clustering methods. During query time, the system will then expand one query token to include a set of its similar tokens and do instance lookup with this expanded token set. To improve the efficiency of this clustering step, we could compute certain hash values for each token in the document collection, and tokens with the same hash value will be put into the same cluster. In this manner, we then have non-overlapping clusters of similar tokens. In future research, I will explore how to incorporate the outcome of data quality related research into coreference systems to better handle noisy and low-quality data.

9.2.4 Improving Value-based Property Matching with Bootstrapping

One other interesting future work would be to develop a more precise and potential more comprehensive property matching scheme. Our current approach is value-based and determines property comparability based upon the token sets of two properties. However, one risk of this approach is on numerical value predicates. For instance, the two predicates *birth-year* and *death-year* could have very similar value spaces but are not comparable given their actual semantics.

To alleviate such potential risk, rather than adopting this value-based approach, we could try bootstrapping the entire coreference process. In general, instead of generating all property comparability in one pre-processing step, we could gradually add more mappings between properties during coreference. These new property pairs are selected from instance pairs having high coreference score and need to be highly similar on their values for such

instance pairs. Suppose we have some initial mappings between predicates. At the beginning of the process, we utilize these mappings to compute the coreference relationships between instances, and pick up the highly similar instance pairs. For such pairs, we figure out the property pairs whose values are highly similar, and such property pairs receive a vote. We finally choose and add those property pairs that receive the most votes into our property mapping pool, and employ these updated mappings in future iterations. These newly added property mappings are expected to improve recall (by enabling the system to match more things) and precision (by allowing the system to apply more appropriate penalties) as well. The entire coreference process will stop when some stopping criteria are met (e.g., no more new property mappings can be found for a given number of iterations).

Take the *birth-year* and *death-year* example again. For highly similar instance pairs, we would not expect that their values of the two properties to be similar for a significant amount of times. Even if this property pair might receive votes due to some potential data errors, it would not happen frequently. Eventually, these two properties, though having highly overlapping value spaces, will not be considered comparable. One potential problem of this bootstrapping approach is that it might take longer to finish, given that it will run iteratively. Until we conduct actual experiments, it might be difficult to say whether its longer runtime would be compensated well with higher F1-score.

9.2.5 Iterative Entity Coreference with Context Merging

As discussed earlier, the proposed entity coreference algorithm in Section 3 misses some true matches because of information heterogeneity. For a particular ontology instance, different data sources may only contain partial information of this instance. For example, different academic databases only index a selected number of publications of an author. Therefore,

when we build the contexts for different instances, only information provided within a single data source can be utilized, assuming we do not utilize the existing *owl:sameAs* linkages across those different datasets to collect context. Exploring the web can be one possible solution to alleviate this information heterogeneity problem; however, for large scale datasets, issuing queries to the web can add additional burden and complexity to the system. Such approaches will also largely depend on the hypothesis that search engines can provide good results. Even if this is the case, some human effort is still needed to clean up the results returned by search engines. Aswani et al. [13] tried issuing queries to search engines to check if two person instances are reported to have the same full name; however, they needed to manually design rules to filter out noisy web pages, and to determine if a name is a valid full name.

In future work, it would be interesting to develop iterative entity coreference algorithms in order to utilize the results from previous iterations to facilitate the current matching process. The algorithm can bootstrap in a way that it gradually merges the context information of highly similar instances in order to reduce the negative impact of information heterogeneity. The key point is to improve recall of the system while trying to minimize the possible negative impact on precision. If two instances are similar to a certain degree, then we merge their contexts to form a new and virtual instance as if it were one that exists in the dataset. This new instance will then be more capable of absorbing other coreferent instances that were not able to exceed a similarity threshold due to insufficient context overlap. Such an iterative algorithm is essentially a clustering algorithm but it is different in that it tries to form new instances with more comprehensive context rather than treating clustered instances separately. Traditional entity coreference research has adopted clustering algorithms but they lack this merging process [5, 14].

However, we need to be careful of setting the criteria for context merging. One simple idea is to compute the full transitive closure. After one iteration, we set a threshold and pick all instance pairs whose similarity scores exceed the threshold; we then compute the full transitive closure, and merge all instances included. The problem is that it is easy to pollute the clusters because the negative impact of a single mistaken link could be magnified. How to intelligently determine the merging criteria would be an interesting future work.

9.3 A Vision of the Semantic Web

The World Wide Web has altered the way we share knowledge by lowering the barrier to publishing and accessing documents as part of a global information and knowledge space. The past and current success of search engines (such as Google and Yahoo) may lead people to believe that the Web has already reached its full potential. However, as we discussed at the very beginning of this dissertation, search engines are not able to meet our information needs due to its limitations of keyword matching and the document based Web.

The Semantic Web is suggesting a way of extending the existing web with structure and providing a mechanism to specify formal semantics that are machine-readable and shareable. By organizing the information better on the Web, and by adopting the Linked Data techniques and principles, we are transitioning the current document based web (a global information space of linked documents) to one where both documents and data are linked. As an interlinked data hub, the Semantic Web is really allowing people to aggregate, discover, and reuse information. Within Linked Data, the heavy reasoning or the AI vision of the Semantic Web has been replaced by a networked and user-driven Semantic Web. This new view of the Semantic web will be more lightweight, and geared toward the application of a far less structured and more organic approach to dealing with the complexities of the

diverse data present in the Web.

In order to achieve the goal of transitioning the current document web to an interlinked data hub, one important technique would be data interlinking or entity coreference. The decentralized nature of the Semantic Web allows data publishers to publish their in their own ontologies with their own identifiers. Without being able to appropriately discover the linkages across instances distributedly published in different data sources, the interlinked vision of the Semantic Web would be an illusion. Our research in this dissertation targets minimizing this gap by developing algorithms to interlink and disambiguate ontology instances in different data sources. In addition to trying to achieve high precision and recall for this interlinking process, we also focused on improving the scalability of our algorithm, aiming to handle the current scale of the Semantic Web. Based on the work I (and others) have done, if the research challenges highlighted in Section 9.2 can be adequately addressed, I expect that Semantic Web and the Linked Data idea will enable a significant evolutionary step in leading the Web to its full potential.

Bibliography

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.

[2] C. L. Giles, K. D. Bollacker, and S. Lawrence, "Citeseer: an automatic citation indexing system," in *Proceedings of the third ACM conference on Digital libraries*, ser. DL '98. New York, NY, USA: ACM, 1998, pp. 89–98.

[3] M. Ley, "The DBLP computer science bibliography: Evolution, research issues, perspectives," in *SPIRE*, 2002, pp. 1–10.

[4] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data - the story so far," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, 2009.

[5] A. Bagga and B. Baldwin, "Entity-based cross-document coreferencing using the vector space model," in *COLING-ACL*, 1998, pp. 79–85.

[6] A. Yates and O. Etzioni, "Unsupervised resolution of objects and relations on the web," in *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, 2007, pp. 121–130.

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 1, pp. 1–16, 2007.

[8] Y. Cao, Z. Chen, J. Zhu, P. Yue, C.-Y. Lin, and Y. Yu, "Leveraging unlabeled data to scale blocking for record linkage," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 2011, pp. 2211–2217.

[9] H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, and H. S. Thompson, "When owl: sameas isn't the same: An analysis of identity in linked data," in *9th International Semantic Web Conference (ISWC)*, 2010, pp. 305–320.

[10] A. Qasem, D. A. Dimitrov, and J. Heflin, "Towards scalable information integration with instance coreferences," in *IJCAI Workshop on Information Integration on the Web*, 2009.

[11] Y. Li and J. Heflin, "Using reformulation trees to optimize queries over distributed heterogeneous sources," in *9th International Semantic Web Conference (ISWC)*, 2010, pp. 502–517.

[12] M. Hausenblas, R. Troncy, T. Bürger, and Y. Raimond, "Interlinking multimedia: How to apply linked data principles to multimedia fragments," in *Proceedings of the WWW2009 Workshop on Linked Data on the Web (LDOW)*, 2009.

[13] N. Aswani, K. Bontcheva, and H. Cunningham, "Mining information for instance unification," in *The 5th International Semantic Web Conference (ISWC)*, 2006, pp. 329–342.

[14] C. H. Gooi and J. Allan, "Cross-document coreference on a large scale corpus," in *HLT-NAACL*, 2004, pp. 9–16.

[15] J. Hassell, B. Aleman-Meza, and I. B. Arpinar, "Ontology-driven automatic entity disambiguation in unstructured text," in *5th International Semantic Web Conference (ISWC)*, 2006, pp. 44–57.

[16] D. Song and J. Heflin, "Domain-independent entity coreference in RDF graphs," in *Proceedings of the 19th ACM Conference on Information and Knowledge Management (CIKM)*, 2010, pp. 1821–1824.

[17] H. Glaser, I. Millard, and A. Jaffri, "Rkbexplorer.com: A knowledge driven infrastructure for linked data providers," in *The 5th European Semantic Web Conference (ESWC)*, 2008, pp. 797–801.

[18] D. Song and J. Heflin, "Domain-independent entity coreference for linking ontology instances," *J. Data and Information Quality*, vol. 4, no. 2, p. 7, 2013.

[19] ——, "A pruning based approach for scalable entity coreference," in *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, 2012, pp. 98–103.

[20] ——, "Accuracy vs. speed: Scalable entity coreference on the semantic web with on-the-fly pruning," in *2012 IEEE/WIC/ACM International Conferences on Web Intelligence (Web Intelligence)*, 2012, pp. 66–73.

[21] ——, "Automatically generating data linkages using a domain-independent candidate selection approach," in *International Semantic Web Conference*, 2011, pp. 649–664.

[22] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *Int. J. Hum.-Comput. Stud.*, vol. 43, no. 5-6, pp. 907–928, 1995.

[23] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters (WWW)*, 2004, pp. 74–83.

[24] M. Horridge and S. Bechhofer, "The owl api: A java api for owl ontologies," *Semantic Web*, vol. 2, no. 1, pp. 11–21, 2011.

[25] F. Baader and W. Nutt, "Basic description logics," in *The Description Logic Handbook: Theory, Implementation, and Applications*, 2003, pp. 43–95.

[26] D. Nardi and R. J. Brachman, "An introduction to description logics," in *The Description Logic Handbook: Theory, Implementation, and Applications*, 2003, pp. 1–40.

[27] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker, "Description logic programs: combining logic programs with description logic," in *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, 2003, pp. 48–57.

[28] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008.

[29] E. Alpaydin, *Introduction to machine learning*. MIT press, 2004.

[30] E. Bengtson and D. Roth, "Understanding the value of features for coreference resolution," in *EMNLP 2008, Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2008, pp. 294–303.

[31] M. Recasens and E. H. Hovy, "Blanc: Implementing the rand index for coreference evaluation," *Natural Language Engineering*, vol. 17, no. 4, pp. 485–510, 2011.

[32] H. Lee, M. Recasens, A. X. Chang, M. Surdeanu, and D. Jurafsky, "Joint entity and event coreference resolution across documents," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2012, pp. 489–500.

[33] P. Jindal and D. Roth, "End-to-end coreference resolution for clinical narratives," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[34] W. Hu and Y. Qu, "Falcon-ao: A practical ontology matching system," *Journal of Web Semantics*, vol. 6, no. 3, pp. 237–239, 2008.

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

[36] D. Dey, V. S. Mookerjee, and D. Liu, "Efficient techniques for online record linkage," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 3, pp. 373–387, 2011.

[37] G. S. Mann and D. Yarowsky, "Unsupervised personal name disambiguation," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL*, 2003, pp. 33–40.

[38] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995. [Online]. Available: http://dx.doi.org/10.1023/A:1022627411411

[39] H. Han, C. L. Giles, H. Zha, C. Li, and K. Tsioutsiouliklis, "Two supervised learning approaches for name disambiguation in author citations," in *JCDL*, 2004, pp. 296–305.

[40] M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios, "Tailor: A record linkage tool box," in *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 2002, pp. 17–28.

[41] R. Baxter, P. Christen, and T. Churches, "A comparison of fast blocking methods for record linkage," in *In ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation*, 2003.

[42] M. Michelson and C. A. Knoblock, "Learning blocking schemes for record linkage," in *The Twenty-First National Conference on Artificial Intelligence (AAAI)*, 2006.

[43] S. Yan, D. Lee, M.-Y. Kan, and C. L. Giles, "Adaptive sorted neighborhood methods for efficient record linkage," in *ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, 2007, pp. 185–194.

[44] M. Michelson and C. A. Knoblock, "Creating relational data from unstructured and ungrammatical data sources," *J. Artif. Intell. Res. (JAIR)*, vol. 31, pp. 543–590, 2008.

[45] J. Artiles, J. Gonzalo, and S. Sekine, "Weps 2 evaluation campaign: overview of the web people search clustering task," in *2nd Web People Search Evaluation Workshop (WePS 2009), 18th WWW Conference*, 2009.

[46] V. Guha and A. Garg, "Disambiguating people in search," in *WWW*, 2004.

[47] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "Dbpedia - a crystallization point for the web of data," *J. Web Sem.*, vol. 7, no. 3, pp. 154–165, 2009.

[48] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "Dbpedia: A nucleus for a web of open data," in *The Semantic Web, 6th International Semantic*

Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007 (ISWC/ASWC), 2007, pp. 722–735.

[49] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2008, pp. 1247–1250.

[50] J. Euzenat, A. Ferrara, C. Meilicke, A. Nikolov, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Svb-Zamazal, V. Svtek, and C. Trojahn dos Santos, "Results of the ontology alignment evaluation initiative 2010," in *Proceedings of the 4th International Workshop on Ontology Matching (OM) collocated with the 9th International Semantic Web Conference (ISWC)*, 2010.

[51] D. Yarowsky, "Unsupervised word sense disambiguation rivaling supervised methods," in *ACL*, 1995, pp. 189–196.

[52] X. Zhang and J. Heflin, "Calculating word sense probability distributions for semantic web applications," in *Proceedings of the 4th IEEE International Conference on Semantic Computing (ICSC)*, 2010, pp. 470–477.

[53] M. A. Hernández and S. J. Stolfo, "The merge/purge problem for large databases," in *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 1995, pp. 127–138.

[54] X. Dong, A. Y. Halevy, and J. Madhavan, "Reference reconciliation in complex information spaces," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005, pp. 85–96.

[55] D. V. Kalashnikov and S. Mehrotra, "Domain-independent data cleaning via analysis of entity-relationship graph," *ACM Trans. Database Syst.*, vol. 31, no. 2, pp. 716–767, 2006.

[56] I. Bhattacharya and L. Getoor, "Collective entity resolution in relational data," *IEEE Data Eng. Bull.*, vol. 29, no. 2, pp. 4–12, 2006.

[57] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis, "On-the-fly entity-aware query processing in the presence of linkage," *PVLDB*, vol. 3, no. 1, pp. 429–438, 2010.

[58] Z. Harris, "Distributional structure," *Word*, vol. 10, no. 23, pp. 146–162, 1954.

[59] T. Pedersen, A. Purandare, and A. Kulkarni, "Name discrimination by clustering similar contexts," in *6th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, 2005, pp. 226–237.

[60] V. Hatzivassiloglou, P. A. Duboue;, and A. Rzhetsky, "Disambiguating proteins, genes, and rna in text: a machine learning approach," *Bioinformatics*, vol. 17 Suppl 1, pp. S97–106, 2001.

[61] B. Baldwin, T. Morton, A. Bagga, J. Baldridge, R. Ch, A. Dimitriadis, K. Snyder, and M. Wolska, "Description of the upenn camp system as used for coreference," in *Proceedings MUC-7*, 1998.

[62] Y. Chen and J. Martin, "Towards robust unsupervised personal name disambiguation," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007, pp. 190–198.

[63] R. Bekkerman and A. McCallum, "Disambiguating web appearances of people in a social network," in *Proceedings of the 14th international conference on World Wide Web (WWW)*, 2005, pp. 463–470.

[64] E. Minkov, W. W. Cohen, and A. Y. Ng, "Contextual search and name disambiguation in email using graphs," in *SIGIR*, 2006, pp. 27–34.

[65] R. C. Bunescu and M. Pasca, "Using encyclopedic knowledge for named entity disambiguation," in *EACL*, 2006.

[66] S. Cucerzan, "Large-scale named entity disambiguation based on Wikipedia data," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 708–716.

[67] N. David and S. Satoshi, "A survey of named entity recognition and classification," *Linguisticae Investigationes*, vol. 30, no. 1, pp. 3–26, January 2007.

[68] B. P. Nunes, S. Dietze, M. A. Casanova, R. Kawase, B. Fetahu, and W. Nejdl, "Combining a co-occurrence-based and a semantic measure for entity linking," in *The Semantic Web: Semantics and Big Data, 10th International Conference (ESWC)*, 2013, pp. 548–562.

[69] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer, "Dbpedia spotlight: shedding light on the web of documents," in *Proceedings the 7th International Conference on Semantic Systems (I-SEMANTICS)*, 2011, pp. 1–8.

[70] X. Han and J. Zhao, "Named entity disambiguation by leveraging wikipedia semantic knowledge," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, 2009, pp. 215–224.

[71] H. Alani, S. Dasmahapatra, N. Gibbins, H. Glaser, S. Harris, Y. Kalfoglou, K. O'Hara, and N. Shadbolt, "Managing reference: Ensuring referential integrity of ontologies for the semantic web," in *EKAW*. Springer, 2002, pp. 317–334.

[72] E. Jiménez-Ruiz and B. C. Grau, "Logmap: Logic-based and scalable ontology matching," in *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference (ISWC), Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, 2011, pp. 273–288.

[73] E. Jiménez-Ruiz, B. C. Grau, Y. Zhou, and I. Horrocks, "Large-scale interactive ontology matching: Algorithms and implementation," in *20th European Conference on Artificial Intelligence (ECAI). Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France*, 2012, pp. 444–449.

[74] W. Hu, J. Chen, and Y. Qu, "A self-training approach for resolving object coreference on the semantic web," in *Proceedings of the 20th International Conference on World Wide Web (WWW)*, 2011, pp. 87–96.

[75] X. Niu, X. Sun, H. Wang, S. Rong, G. Qi, and Y. Yu, "Zhishi.me - weaving Chinese linking open data," in *International Semantic Web Conference*, 2011, pp. 205–220.

[76] F. Saïs, N. Pernelle, and M.-C. Rousset, "Combining a logical and a numerical method for data reconciliation," *Journal on Data Semantics XII*, vol. 12, pp. 66–94, 2009.

[77] J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt, "Leveraging terminological structure for object reconciliation," in *7th Extended Semantic Web Conference (ESWC)*, 2010, pp. 334–348.

[78] Y. R. Jean-Mary, E. P. Shironoshita, and M. R. Kabuka, "Ontology matching with semantic verification," *Journal of Web Semantics*, vol. 7, pp. 235–251, September 2009.

[79] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[80] C. Sarasua, E. Simperl, and N. F. Noy, "Crowdmap: Crowdsourcing ontology alignment with microtasks," in *11th International Semantic Web Conference (ISWC)*, 2012, pp. 525–541.

[81] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux, "Large-scale linked data integration using probabilistic reasoning and crowdsourcing," *VLDB J.*, vol. 22, no. 5, pp. 665–687, 2013.

[82] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1348–1358, 2010.

[83] D. E. Knuth, *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[84] R. L. Taft, "Name search techniques," New York State Identification and Intelligence System, Albany, NY, Tech. Rep. Special Report No. 1, February 1970.

[85] H. B. Newcombe and J. M. Kennedy, "Record linkage: making maximum use of the discriminating power of identifying information," *Commun. ACM*, vol. 5, no. 11, pp. 563–566, 1962.

[86] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, 2000, pp. 169–178.

[87] W. E. Winkler, "Approximate string comparator search strategies for very large administrative lists," Statistical Research Division, U.S. Census Bureau, Tech. Rep., 2005.

[88] M. A. Hernández and S. J. Stolfo, "Real-world data is dirty: Data cleansing and the merge/purge problem," *Data Min. Knowl. Discov.*, vol. 2, no. 1, pp. 9–37, 1998.

[89] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov, "Discovering and maintaining links on the web of data," in *8th International Semantic Web Conference (ISWC)*, 2009, pp. 650–665.

[90] J. R. Talburt, *Entity Resolution and Information Quality*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[91] L. Gu and R. A. Baxter, "Adaptive filtering for efficient record linkage," in *Proceedings of the Fourth SIAM International Conference on Data Mining*, 2004.

[92] M. Bilenko and R. J. Mooney, "Adaptive duplicate detection using learnable string similarity measures," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 39–48.

[93] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, 2006, pp. 918–929.

[94] J. Wang, G. Li, and J. Feng, "Trie-join: Efficient trie-based string similarity joins with edit-distance constraints," *PVLDB*, vol. 3, no. 1, pp. 1219–1230, 2010.

[95] J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin, "Efficient exact edit similarity query processing with the asymmetric signature scheme," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2011, pp. 1033–1044.

[96] J. Wang, G. Li, and J. Feng, "Fast-join: An efficient method for fuzzy token matching based string similarity join," in *Proceedings of the 27th International Conference on Data Engineering (ICDE)*, 2011, pp. 458–469.

[97] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proceedings of the 16th International Conference on World Wide Web (WWW)*, 2007, pp. 131–140.

[98] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *Proceedings of the 17th International Conference on World Wide Web (WWW)*, 2008, pp. 131–140.

[99] C. Xiao, W. Wang, and X. Lin, "Ed-join: an efficient algorithm for similarity joins with edit distance constraints," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 933–944, 2008.

[100] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang, "Efficient similarity joins for near-duplicate detection," *ACM Trans. Database Syst.*, vol. 36, no. 3, p. 15, 2011.

[101] E. Ioannou, O. Papapetrou, D. Skoutas, and W. Nejdl, "Efficient semantic-aware detection of near duplicate resources," in *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference (ESWC)*, 2010, pp. 136–150.

[102] D. Koller and M. Sahami, "Hierarchically classifying documents using very few words," in *Proceedings of the Fourteenth International Conference on Machine Learning (ICML)*, 1997, pp. 170–178.

[103] D. Song and J. Heflin, "Domain-independent entity coreference for linking ontology instances," *ACM Journal of Data and Information Quality (ACM JDIQ)*, vol. 1, no. 1, p. Accepted, 2012.

[104] D. D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *10th European Conference on Machine Learning (ECML)*, 1998, pp. 4–15.

[105] N. F. Noy and M. A. Musen, "Prompt: Algorithm and tool for automated ontology merging and alignment," in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, 2000, pp. 450–455.

[106] J. Euzenat, "An api for ontology alignment," in *International Semantic Web Conference*, 2004, pp. 698–712.

[107] S. Pavel and J. Euzenat, "Ontology matching: State of the art and future challenges," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 99, p. 99, 2011.

[108] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, 2003, pp. 73–78.

[109] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[110] P. Christen, "Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008, pp. 1065–1068.

[111] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

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

[113] A. Harth, "Billion Triples Challenge data set," Downloaded from http://km.aifb.kit.edu/projects/btc-2012/, 2012.

[114] G. Stoilos, G. B. Stamou, and S. D. Kollias, "A string metric for ontology alignment," in *International Semantic Web Conference (ISWC)*, 2005, pp. 624–637.

[115] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443 – 453, 1970.

[116] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage," in *Proceedings of the Section on Survey Research*, 1990, pp. 354–359.

[117] Y. R. Jean-Mary, E. P. Shironoshita, and M. R. Kabuka, "Ontology matching with semantic verification," *J. Web Sem.*, vol. 7, no. 3, pp. 235–251, 2009.

[118] M. Nagy, M. Vargas-Vera, and P. Stolarski, "Dssim results for oaei 2009," in *Proceedings of the 4th International Workshop on Ontology Matching collocated with the 8th International Semantic Web Conference*, 2009.

[119] W.-S. Li and C. Clifton, "Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural networks," *Data Knowl. Eng.*, vol. 33, no. 1, pp. 49–84, 2000.

[120] Q. Y. Wang, J. X. Yu, and K.-F. Wong, "Approximate graph schema extraction for semi-structured data," in *7th International Conference on Extending Database Technology, March 27-31, 2000, Proceedings*, C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, Eds., vol. 1777. Springer, 2000, pp. 302–316.

[121] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.

[122] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *J. ACM*, vol. 22, no. 2, pp. 215–225, 1975.

[123] R. E. Tarjan and J. van Leeuwen, "Worst-case analysis of set union algorithms," *J. ACM*, vol. 31, no. 2, pp. 245–281, 1984.

[124] S. Araújo, D. Tran, A. DeVries, J. Hidders, and D. Schwabe, "Serimi: Class-based disambiguation for effective instance matching over heterogeneous web data," in *Proceedings of the 15th International Workshop on the Web and Databases (WebDB), Scottsdale, AZ, USA, May 20, 2012*, 2012, pp. 25–30.

[125] R. Sankaranarayanan, L. Gaffikin, M. Jacob, J. Sellors, and S. Robles, "A critical assessment of screening methods for cervical neoplasia," *International Journal of Gynecology and Obstetrics*, vol. 89, pp. 4–12, 2005.

[126] C. Ferreccio, M. I. Barriga, M. Lagos, C. Ibez, H. Poggi, F. Gonzlez, S. Terrazas, H. A. Katki, F. Nez, J. Cartagena, V. Van De Wyngard, D. Viales, and J. Braes, "Screening trial of human papillomavirus for early detection of cervical cancer in santiago, chile," *International Journal of Cancer*, vol. 132, no. 4, pp. 916–923, 2013.

[127] M.-H. Mayrand, E. Duarte-Franco, I. Rodrigues, S. D. Walter, J. Hanley, A. Ferenczy, S. Ratnam, F. Coutl, and E. L. Franco, "Human papillomavirus dna versus papanicolaou screening tests for cervical cancer," *New England Journal of Medicine*, vol. 357, no. 16, pp. 1579–1588, 2007.

[128] C. V. Biscotti, A. E. Dawson, B. Dziura, L. Galup, T. Darragh, A. Rahemtulla, and L. Wills-Frank, "Assisted primary screening using the automated thinprep imaging system," *American Journal of Clinical Pathology*, vol. 123, no. 2, pp. 281–287, 2005.

[129] D. C. Wilbur, W. S. Black-Schaffer, R. D. Luff, K. P. Abraham, C. Kemper, J. T. Molina, and W. D. Tench, "The becton dickinson focalpoint gs imaging system: Clinical trials demonstrate significantly improved sensitivity for the detection of important cervical lesions," *American Journal of Clinical Pathology*, vol. 132, no. 5, pp. 767–775, 2009.

[130] E. P. Whitlock, K. K. Vesco, M. Eder, J. S. Lin, C. A. Senger, and B. U. Burda, "Liquid-based cytology and human papillomavirus testing to screen for cervical cancer: A systematic review for the u.s. preventive services task force," *Annals of Internal Medicine*, vol. 155, no. 10, pp. 687–697, 2011.

[131] S. AG, K. PM, G. JM, and et al, "Comparison of liquid-based cytology with conventional cytology for detection of cervical cancer precursors: A randomized controlled trial," *JAMA*, vol. 302, no. 16, pp. 1757–1764, 2009.

[132] K. Hartmann, S. Hall, K. Nanda, and et al., "Screening for cervical cancer: Systematic evidence review," Agency for Healthcare Research and Quality: United States Preventive Service Task Force, 2002.

[133] Y. Srinivasana, D. Hernesa, B. Tulpulea, S. Yanga, J. Guoa, S. Mitraa, S. Yagneswarana, B. Nuttera, J. Jeronimob, B. Phillipsc, R. Long, and D. Ferris, "A multi-spectral digital cervigramtm analyzer in the wavelet domain for early detection of cervical cancer," in *Proc. of SPIE, Medical Imaging: Image Processing*, vol. 5747, 2005.

[134] D. Ferris, M. Schiffman, and L. M.S., "Cervicography for triage of women with mildly abnormal cervical cytology results," *American Journal of Obstetrics and Gynecology*, vol. 185, no. 4, pp. 939–43, 2001.

[135] C. Eskridge, W. Begneaud, and C. Landwehr, "Cervicography combined with repeated papanicolaou test as triage for low-grade cytologic abnormalities," *Obstetrics and Gynecology*, vol. 92, no. 3, pp. 351–355, 1998.

[136] L. S. Massad, J. Jeronimo, H. A. Katki, and M. Schiffman, "The accuracy of colposcopic grading for detection of high-grade cervical intraepithelial neoplasia," *Journal of Lower Genital Tract Disease*, vol. 13, no. 3, pp. 137–144, 2009.

[137] J. Jeronimo, L. R. Long, L. Neve, B. Michael, S. Antani, and M. Schiffman, "Digital tools for collecting data from cervigrams for research and training in colposcopy." *Journal of Lower Genital Tract Disease*, vol. 10, no. 1, pp. 16–25, 2006.

[138] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang, "Efficient similarity joins for near-duplicate detection," *ACM Trans. Database Syst.*, vol. 36, no. 3, pp. 15:1–15:41, Aug. 2011.

[139] E. Kim and X. Huang, "A data driven approach to cervigram image analysis and classification," in *Color Medical Image Analysis*, ser. Lecture Notes in Computational Vision and Biomechanics, M. E. Celebi and G. Schaefer, Eds. Springer Netherlands, 2013, vol. 6, pp. 1–13.

[140] D. W. Aha, "Lazy learning: Special issue editorial," *Artificial Intelligence Review*, vol. 11, pp. 1–5, 1997.

[141] M. Batet, D. Sánchez, and A. Valls, "An ontology-based measure to compute semantic similarity in biomedicine," *Journal of Biomedical Informatics*, vol. 44, no. 1, pp. 118–125, 2011.

[142] R. Sankaranarayanan, P. Basu, R. S. Wesley, C. Mahe, N. Keita, C. C. G. Mbalawa, R. Sharma, A. Dolo, S. S. Shastri, M. Nacoulma, M. Nayama, T. Somanathan, E. Lucas, R. Muwonge, L. Frappart, D. M. Parkin, and for the IARC Multicentre Study Group on Cervical Cancer Early Detection, "Accuracy of visual screening for cervical

neoplasia: Results from an iarc multicentre study in india and africa," *International Journal of Cancer*, vol. 110, no. 6, pp. 907–913, 2004.

[143] C. Ferreccio, M. Bratti, M. Sherman, R. Herrero, S. Wacholder, A. Hildesheim, R. Burk, M. Hutchinson, M. Alfaro, M. Greenberg, J. Morales, A. Rodriguez, J. Schussler, C. Eklund, G. Marshall, and M. Schiffman, "A comparison of single and combined visual, cytologic, and virologic tests as screening strategies in a region at high risk of cervical cancer." *Cancer Epidemiol Biomarkers Prev*, vol. 12, no. 9, pp. 815–823, 09 2003.

[144] J. Cuzick, A. Szarewski, H. Cubie, G. Hulman, H. Kitchener, D. Luesley, E. McGoogan, U. Menon, G. Terry, R. Edwards, C. Brooks, M. Desai, C. Gie, L. Ho, I. Jacobs, C. Pickles, and P. Sasieni, "Management of women who test positive for high-risk types of human papillomavirus: the {HART} study," *The Lancet*, vol. 362, no. 9399, pp. 1871 – 1876, 2003.

[145] D. Saslow, D. Solomon, H. W. Lawson, M. Killackey, S. L. Kulasingam, J. Cain, F. A. R. Garcia, A. T. Moriarty, A. G. Waxman, D. C. Wilbur, N. Wentzensen, L. S. Downs, M. Spitzer, A.-B. Moscicki, E. L. Franco, M. H. Stoler, M. Schiffman, P. E. Castle, E. R. Myers, and A.-A.-A. C. C. G. Committee, "American cancer society, american society for colposcopy and cervical pathology, and american society for clinical pathology screening guidelines for the prevention and early detection of cervical cancer," *CA: A Cancer Journal for Clinicians*, vol. 62, no. 3, pp. 147–172, 2012.

[146] S. Khatchadourian and M. P. Consens, "Understanding billions of triples with usage summaries," in *Semantic Web Challenge*, 2011.

[147] Y. Yu and J. Heflin, "Extending functional dependency to detect abnormal data in rdf graphs," in *International Semantic Web Conference (1)*, 2011, pp. 794–809.

[148] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and efficient fuzzy match for online data cleaning," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2003, pp. 313–324.

Vita

Born in Beijing, the capital city of China, Dezhao Song received both his Undergraduate and Master Degrees in the School of Computer Science and Engineering at Beijing Institute of Technology, Beijing, China. He was awarded scholarship from the School of Computer Science and Engineering at Beijing Institute of Technology several times for academic excellence. At Lehigh University, Bethlehem, PA, USA, he has been both a teaching and research assistant for the Department of Computer Science and Engineering while completing his Ph.D. During the summer of the year of 2011 and 2012, he was research intern at Mayo Clinic and IBM Ireland Research Center. Dezhao's research focuses on developing novel algorithms in integrating heterogeneous and large-scale data in the Semantic Web, targeting the goal of building an interlinked data hub for the Semantic Web and fulfilling the goal of Linked Data. After obtaining his Ph.D. in Computer Science from Lehigh University, he will be joining Thomson Reuters as a Research Scientist in Eagan, Minnesota.