Reconcile: A Coreference Resolution Research Platform

Veselin Stoyanov
Claire Cardie
Dept. of Computer Science
Cornell University
Ithaca, NY
ves@cs.cornell.edu
cardie@cs.cornell.edu

Nathan Gilbert
Ellen Riloff
School of Computing
University of Utah
Salt Lake City, UT
ngilbert@cs.utah.edu
riloff@cs.utah.edu

David Buttler
David Hysom
Lawrence Livermore
National Laboratory
Livermore, CA
buttler1@llnl.gov
hysom1@llnl.gov

Abstract

We have created a software infrastructure called Reconcile that is a platform for the development of learning-based noun phrase (NP) coreference resolution systems. Reconcile's architecture was designed to facilitate the rapid creation of coreference resolutions systems, easy implementation of new feature sets and approaches to coreference resolution, and empirical evaluation of coreference resolvers across a variety of benchmark data sets and standard scoring metrics. Reconcile is written in Java and can be easily customized with different subcomponents, feature sets, and parameter settings. In this report, we describe Reconcile's architecture, processing pipeline, and the subcomponents and algorithms that are currently implemented and available in Reconcile. We also present experimental results showing that Reconcile can be used to create a coreference resolver which achieves performance levels comparable to state-of-the-art systems on six benchmark data sets.

1 Introduction

Noun phrase coreference resolution (or simply coreference resolution) is the problem of identifying all noun phrases (NPs) that refer to the same real-world entity in text. Coreference resolution is one of the fundamental problems of natural language processing (NLP) because of its usefulness for other NLP tasks (e.g. Morton (1999), Steinberger et al. (2007)) as well as the theoretical interest in understanding computational mechanisms involved in government, binding and linguistic reference. Figure 1 displays an example of coreferent noun phrases from a sample news article.

Several formal evaluations have been conducted for the coreference resolution task, and the data sets created for this evaluations have become standard benchmarks in the field (e.g., MUC and ACE). However, it is still frustratingly difficult to compare results across different coreference resolution systems. Reported scores for coreference resolution vary wildly across data sets, evaluation metrics, and system configurations.

We believe that one root cause of these disparities is the high cost of implementing an end-to-end coreference resolution system. Coreference resolution is a complex problem, and successful systems must tackle a variety of non-trivial subproblems, which themselves require substantial implementation efforts. As a result, many researchers exploit gold-standard annotations from available data sets as a substitute for component technologies to solve these subproblems. Many published research results use gold standard annotations to identify the NPs (substituting for mention/markable detection), to distinguish the anaphoric NPs from the non-anaphoric NPs (substituting for anaphoricity determination), to identify named entities (substituting for named entity recognition), and to identify semantic types of nominals (substituting for semantic class identification). The use of gold standard annotations leads to an unrealistic evaluation setting, and makes it impossible to directly compare results against coreference resolvers that solve all of these subproblems from scratch.

The problem is further magnified by the availability of several (non-trivial) evaluation measures, and data sets that have substantially different task definitions and annotation formats. Understandably, research efforts rarely implement all of the evaluation metrics or evaluate their systems on all of the data sets. As a result, even when different approaches use the same system configuration, they may conduct experiments on different data sets or use different evaluation metrics, which again makes it impossible to directly compare results.

In addition, coreference is a pervasive prob-

A state security court suspended a newspaper$_1$ critical of the government Saturday after convicting it$_2$ of publishing religiously inflammatory material, the newspaper$_3$'s lawyer$_1$ said.

The Khartoum Monitor$_4$ may not appear for two months and must pay a fine$_1$ of 1 million Sudanese pounds (about US$400), lawyer Ngor Olang Ngor$_2$ told The Associated Press. Failure to pay the fine$_2$ would result in a two-month extension of the suspension.

The sentence was the latest in a series of state actions against the Monitor$_5$, the only English language daily in Sudan$_6$ and a leading critic$_7$ of conditions in the south of the country, where a civil war has been waged for 20 years.

Ngor$_3$ said the paper$_8$ would pay the penalty$_3$ but it$_9$ would also appeal the verdict.

He$_4$ said the court ruling stated that the verdict and sentence would be submitted to the National Press Council, a government-appointed body, for approval.

Figure 1: Example noun phrase coreference chains from a sample news article.

lem in natural language processing, so in principle nearly any NLP application could benefit from having an effective coreference resolver. But the hefty price of creating a coreference resolver from scratch often prohibits both researchers and NLP practitioners from incorporating coreference resolution in larger application systems.

To address these issues, we have created a platform for coreference resolution, called Reconcile, that can serve as a software infrastructure to support the creation, experimentation, and evaluation of learning-based coreference resolvers. Our goal in creating Reconcile was to build a coreference resolution system that would allow for different component technologies to be easily swapped in and out (e.g., different parsers, NER systems, or classification algorithms) and for new features to be easily created and plugged in. We also wanted a platform that would provide built-in capabilities to process many of the standard benchmark data sets, respecting their idiosyncratic formats and task definitions, as well as evaluation tools to produce scores that conform to multiple widely used evaluation metrics. This software infrastructure will allow researchers to focus on the new ideas and methodological improvements that they wish to investigate, without having to invest the substantial time and effort required to implement the various subcomponents themselves. Furthermore, this platform will allow researchers to more easily evaluate the impact of their ideas in a complete end-to-end coreference resolution system, with respect to multiple standard data sets and evaluation metrics. We hope that this software infrastructure will help to advance the state-of-the-art in conference resolution by enabling researchers to easily produce results that are directly comparable, yielding better insights into which ideas and techniques perform the best.

This technical report describes the Reconcile coreference resolution platform. Reconcile was designed with the following six desiderata in mind:

- implement the basic underlying software architecture of contemporary state-of-the-art learning-based coreference resolution systems.
- support text analysis on most of the standard coreference resolution data sets.
- implement many widely used coreference resolution scoring metrics.
- exhibit state-of-the-art coreference resolution performance (i.e., it can be configured to create a coreference resolver that achieves performance levels that are close to the best results previously reported in the literature).
- can be extended with new methods and features with little effort.

- is relatively fast and easy to configure and run.

While several other coreference systems are publicly available (e.g., Poesio and Kabadjov (2004), Qiu et al. (2004) and Versley et al. (2008)), none of them meets all six of these requirements (see Related Work section for more details). Reconcile is a modular software platform that abstracts the basic architecture of the most contemporary supervised learning-based coreference resolution systems (e.g. Soon et al. (2001), Ng and Cardie (2002), McCallum and Wellner (2004), Ng (2008)) and achieves performance comparable to the current state-of-the-art on several benchmark data sets commonly used to evaluate coreference resolution systems. Additionally, due to its modularity, Reconcile can be easily reconfigured to use different algorithms, features, preprocessing elements, evaluation settings and metrics.

This report explains how Reconcile is organized, what it can do, and how it performs on several MUC and ACE data sets. Section 2 first explains how Reconcile differs from other publicly available coreference resolution software systems. Section 3 presents Reconcile's organization and describes its subcomponents. Finally, Section 4 shows experimental results for Reconcile on a variety of different data sets and with several evaluation metrics.

2 Related Work

Several coreference resolution systems are currently publicly available. JavaRap (Qiu et al., 2004) is an implementation of the Lappin and Leass's (1994) Resolution of Anaphora Procedure (RAP). JavaRap resolves only pronouns and, thus, it is not directly comparable to Reconcile. GuiTAR (Poesio and Kabadjov, 2004) and BART (Versley et al., 2008) (which can be considered a successor of GuiTar) are both modular systems that target the full coreference resolution task. As such, both of these systems meet the majority of the desiderata set forth in Section 1. BART can be considered an alternative to Reconcile, although we believe that Reconcile's approach is more flexible than BART. The architecture and system components of Reconcile (including a comprehensive set of implemented features that draw on the expertise of state-of-the-art supervised learning approaches, such as Soon et al. (2001) and Ng and Cardie (2002)) result in performance that is closer to the state-of-the-art.

The problem of coreference resolution has received much research attention, resulting in a rich variety of approaches, algorithms and features. Reconcile's architecture has been modeled after the coreference system of Soon et al. (2001) because of its simplicity and its proven competitive performance. In addition, Reconcile includes in its base distribution the refined and extended feature set of Ng and Cardie (2002).

Reconcile has been modeled after typical supervised learning approaches to coreference resolution because of the popularity and relatively good performance of these systems. However, there have been many other approaches to coreference resolution, including recent unsupervised and semi-supervised approaches (e.g. Ng (2008) and Haghighi and Klein (2007)), structured approaches (e.g. McCallum and Wellner (2004) and Finley and Joachims (2005)), competition approaches (e.g. Yang et al. (2003)) and a bell-tree search approach Luo et al. (2004).

As most of these approaches rely on some notion of pairwise feature-based similarity, they can be directly implemented in Reconcile, while some (such as the latter two) require more serious modifications to the system, but can, nevertheless, be run in the system and can benefit from the utilities included in Reconcile that are geared toward handling NPs.

In Section 4, we position Reconcile's performance with respect to the state-of-the-art by comparing Reconcile's results on several standard data sets with the most comparable state-of-the-art results reported in the literature.

3 System Description

3.1 Overall Architecture

Reconcile was designed to be a research test-bed capable of implementing most current state-of-the-art approaches to coreference resolution. Reconcile is written in Java, to be easily portable across computing platforms, and was designed to be easily reconfigurable with respect to subcomponents, feature sets, parameter settings. etc.

The basic architecture of the system includes five major steps, which are performed by most coreference resolution systems (e.g., (Soon et al., 2001), (Ng and Cardie, 2002), (Ng, 2008), (Haghighi and Klein, 2007), (McCallum and Well-

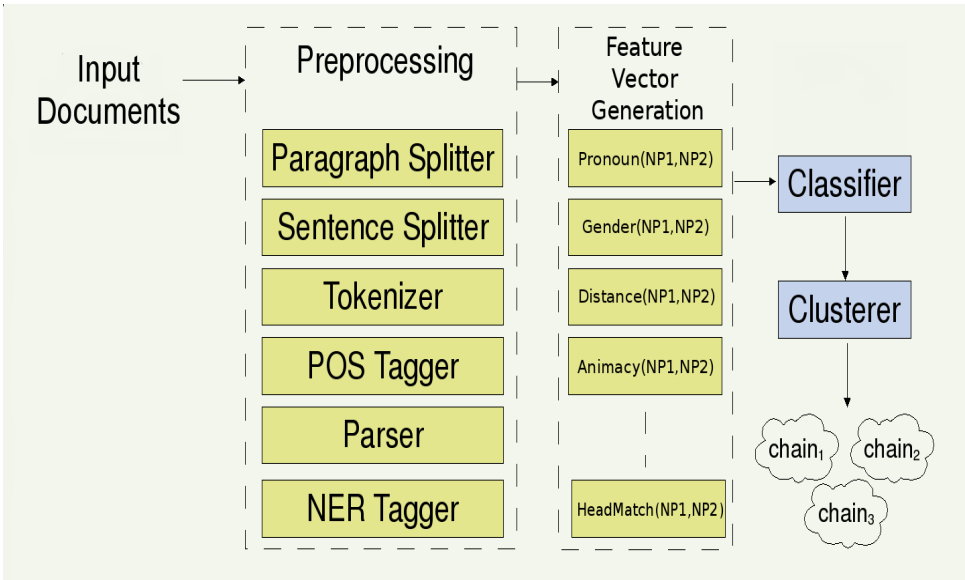

Figure 2: Diagram of the Reconcile classification architecture.

ner, 2004) and (Finley and Joachims, 2005)). Starting with a corpus of text documents together with a manually annotated coreference resolution answer key[1], Reconcile performs the following 5 steps, in order:

1. **Preprocessing.** All documents are passed through a series of (external) linguistic processors such as tokenizers, part-of-speach taggers, syntactic parsers, etc. These components produce annotations of the text in the documents.

2. **Feature generation.** Using the annotations produced during preprocessing, Reconcile creates a set of features. Each feature is an attribute that characterizes a pair of noun phrases. For example, a feature might denote whether two noun phrases agree in number, or whether two noun phrases share any words in common.

3. **Classification** Since Reconcile adopts a supervised learning framework, the system can be used both to train a classifier (using training texts paired with their answer keys) and to apply the classifier to new text documents. The features are used to create one feature vector for each pair of noun phrases. When performing coreference resolution, the classifier is given the feature vector representing a noun phrase pair and it returns a score indicating how likely it is that the CEs are coreferent.

4. **Clustering.** Based on the scores output by the classifier, Reconcile uses a clustering algorithm to form the final set of coreference clusters (chains).

5. **Scoring.** Finally, Reconcile runs scoring algorithms that compare the coreference chains produced by the system to the gold standard coreference chains in the answer key. (This is only possible if the documents have corresponding answer keys.)

Note that some structured coreference resolution algorithms (e.g., (McCallum and Wellner, 2004) and (Finley and Joachims, 2005)) combine the classification and clustering steps above. Reconcile can easily accommodate this modification.

Reconcile's architecture is illstrated in Figure 2. For simplicity, Figure 2 shows Reconcile's operation during the classification phrase (i.e., assuming that a trained classifier is present).

Each of the five major steps described above can include multiple substeps and invoke several components. Reconcile allows various subsystems to be introduced or removed according to the desires of the user. Reconcile's standard distribu-

[1]When Reconcile is applied to texts to perform coreference resolution, Reconcile does not require the presence of a manually annotated answer key. For the purposes of this report, however, we will assume that Reconcile is being used both to create a coreference resolver and to evaluate its performance on standard data sets, which requires answer keys

tion comes with a comprehensive set of implemented components. Additionally, the user is allowed to implement and introduce her own subsystems/components by using some simple API's. Next, we describe the set of implemented components for each of the five steps that come prepackaged in the standard Reconcile distribution, giving more details of the steps in the process.

3.2 Component Subsystems

One of the primary goals in developing Reconcile was to support a plug-and-play architecture where different components could be easily swapped in and out. Toward this end, we have included several different software options for many of the subtasks that Reconcile performs. The components that are packaged in the current release of Reconcile are described below, but it should also be easy for anyone to plug in their own favorite component to perform any of these tasks.

3.2.1 Preprocessing

Reconcile performs a series of preprocessing steps and supports a variety of different preprocessing tools:

Sentence Splitting: Reconcile offers two different sentence segmentation options: The University of Illinois Urbana-Champaign sentence segmentation tool (Cognitive Computation Group, 2009), and the OpenNLP project's sentence splitter (Baldridge, J., 2005). The UIUC segmentation tool is a very fast, heuristic approach. The OpenNLP sentence detector is based on a maximum entropy classifier with a standard English model that can be retrained by the user if desired.

Tokenizing: The tokenizer is responsible for identifying word boundaries based on whitespace and punctuation to produce tokens that are used by the downstream components, such as parsers and part of speech taggers. The tokenizer included in the Reconcile distribution is the OpenNLP project's maximum entropy tokenizer with a standard English model (Baldridge, J., 2005).

Part of Speech Tagging: The Reconcile distribution includes the OpenNLP maximum entropy part-of-speech (POS) tagger. Additionally, POS tags are produced by the two parsers listed below, so POS tags could be easily obtained from the parsers (Baldridge, J., 2005).

Parsing: Reconcile currently includes two probabilistic context-free grammar (PCFG) full parsers, the Stanford (Klein and Manning, 2003) parser and the Berkeley (Petrov and Klein, 2007) parser. The standard models provided with each of these parsers have been trained on Wall Street Journal data.

Named Entity Recognition: Reconcile supports two named entity recognizers: the OpenNLP Maximum-Entropy NER system (Baldridge, J., 2005), and the Stanford Conditional Random Field NER system (Finkel et al., 2004). All available models for both systems have been trained on Wall Street Journal texts. In addition, in-house trained models are provided to improve date and time recognitions.

Coreference Element Identification: The MUC and ACE evaluations used different criteria to define the set of NPs[2] that can participate in the coreference relation. For the sake of generality, we will use the term **coreference element (CE)** to refer to the set of linguistic expressions that participate in the coreference relation. We define the term CE to be roughly equivalent to (a) the notion of *markable* with respect to the MUC task definition and (b) the structures that can be *mentions* with respect to the ACE task definition.

Reconcile includes CE extractors that conform to both the MUC and ACE specifications. These CE extractors were developed in-house and rely on both parsing and named entity recognition.[3] At a high level, both the MUC and ACE evaluations define NPs as nouns, pronouns, and noun phrases. However, the MUC definition excludes (1) "nested" named entities (NEs) (e.g. "America" in "Bank of America"), (2) relative pronouns, and (3) gerunds, but allows (4) nested nouns (e.g. "union" in "union members"). The ACE definition, on the other hand, includes relative pronouns and gerunds, excludes **all** nested nouns that are not themselves CEs, and allows premodifier NE mentions of geo-political entities and locations, such as "Russian" in "Russian politicians".

Another major distinction is that ACE restricts CEs to be entities belonging to one of seven semantic classes: person, organization, geo-political entity, location, facility, vehicle, and weapon. MUC has no semantic restrictions.

[2] Strictly speaking, both the MUC and ACE evaluations define NP coreference resolution over structures that are not always NPs, but both task definitions are generally described as NP coreference resolution.

[3] Our CE extractors were modeled after the extractors initially developed by Ng and Cardie (2002).

```
procedure ExtractCorefElements(D, NPs, NEs){
A text document D
NPs := all noun phrases in D.
NEs := the set of named entities in D.

/* Extract base NPs -- those that do not contain embedded clauses
and prepositions and are not nested */
initialize bNPs := NPs
foreach noun phrase np ∈ bNPs do
if np contains a preposition or embedded clause:
  Remove np from bNPs
if np is the child of another np in bNPs:
  Remove np from bNPs

/* Consolidate NPs and NEs */
foreach noun phrase np ∈ bNPs do
oNEs is the set of those NEs that overlap np in span
  foreach named entity ne ∈ oNEs do
    if neither np covers ne nor ne covers np:
      Expand np's bytespan so that it subsumes ne.
      Remove ne from NEs
    else if ne is a proper substring of np and the two have the same head:
      remove ne to NEs

initialize resultNPs := bNPs
/* Add in named entities that are not part of the NP set.  */
  Add ne to resultNPs

/* Add in nested noun phrases that are not already in the set.  */
foreach noun phrase np ∈ resultNPs do
  if np contains a nested NP from NPs -- nnp ∉ resultNPs
  and resultNPs does not contain a NP with the same head:
    Add nnp to NPs
do the same for nested nouns (non-NPs) when applicable
return NPs
}
```

Figure 3: The algorithm for generating *coreference elements*.

Figure 3 shows the detailed algorithm that Reconcile uses to collect all *coreference elements* from a document.

3.2.2 Classification

Most supervised learning approaches for coreference resolution rely on a pairwise function to determine whether or not two CEs are coreferent. A classifier is trained on CE pairs that are labeled as either *coreferent* or *not coreferent*, and can then be applied to new CE pairs to determine whether that pair of CEs is coreferent. Reconcile adopts this pairwise classification framework.

To train a classifier, Reconcile creates a feature vector for every pair of CEs in a document. Therefore, given a document that contains n coreference elements, the total number of feature vectors created will be $\frac{n^2-n}{2}$. Once the classifier has been trained, it can be given pairwise feature vectors for unseen documents and it will output a probability indicating how likely it is that the two CEs are coreferent.

We developed an extensive set of features, based largely on the feature sets that have been used by other successful coreference resolution systems, most notably the Soon et al. (Soon et al., 2001) and Cardie & Ng (Soon et al., 2001) systems. A comprehensive list and description of the 88 features that are currently available in Reconcile can be found in the Appendix. It should be noted that some of the features rely on properties of individual CEs – for example, the Gender feature has a value of 'Y' if the two CEs in the pair have the same gender. Features that look at individual CEs are cached so they are not recomputed every time a CE participates in a pair.

The current Reconcile release supports a wide variety of learning algorithms. Most of these learning algorithms are included via the Weka toolkit, which is interfaced and included as part of the Reconcile distribution. Additionally, Reconcile includes interfaces for the libSVM (Chang and Lin, 2001) package and SVM$_{light}$ (Joachims, 2002), both of which are implementations of Support Vector Machines (SVMs).

3.2.3 Clustering Algorithms

The last step of the coreference resolution process is clustering. The goal of the clustering algorithm is to consolidate the pairwise decisions of the classifiers and produce a coherent clustering of the CEs in a document. Three different clustering algorithms are implemented in Reconcile:

1. Single-link Clustering (Transitive Closure):

This is a simple clustering algorithm that performs transitive closure of the clustering relation. In other words, the single-link clusterer groups together all CEs that are connected by a path of coreferent links.

2. Best-First: Most classifiers return a value that can be interpreted as a "confidence measure." For instance, an SVM may return a distance from the hyperplane for each pair. Best-first clustering uses the classifier's confidence value to cluster each noun phrase with its most confident antecedent, given that the value for the most confident antecedent is above a pre-specified threshold.

3. Most Recent First: This clustering algorithm pairs each noun phrase with the single most recent antecedent that is labeled as coreferent. This algorithm follows the intuition that the coreference relation is local.

3.2.4 Scoring

The coreference chains produced by the system can be compared to gold standard answer keys to quantitatively assess their quality. Toward that end, Reconcile implements three widely-used automatic scoring functions, which are described below.

MUC score. The MUC scoring algorithm (Vilain et al., 1995) computes the F_1 score (harmonic mean) of precision and recall based on the identification of unique coreference links.

B^3 score. The B^3 algorithm (Bagga and Baldwin, 1998) computes a precision and recall score for each coreference element (ce) using the following formulas:

$precision(ce) = |R_{ce} \cap K_{ce}|/|R_{ce}|$
$recall(ce) = |R_{ce} \cap K_{ce}|/|K_{ce}|$,

where R_{ce} is the coreference chain to which ce is assigned in the response (i.e. the system-generated output) and K_{ce} is the coreference chain that contains ce in the key (i.e. the gold standard). Precision and recall for a set of documents are computed as the mean over all coreference elements (CEs) in the documents. The final B^3 score is the F_1 (harmonic mean) of precision and recall.

B^3 Implementation Issues. Unlike the MUC score, which counts links between CEs, B^3 presumes that the gold standard and the system response are clusterings over the same set of NPs. This, of course, is not the case when the system automatically identifies the CEs, so the scoring algorithm requires a mapping between extracted and annotated NPs. We will use the term $twin(ce)$ to refer to the unique annotated/extracted NP to which the extracted/annotated NP is matched. We say that a NP is *twinless* (has no twin) if no corresponding NP is identified. A twinless extracted NP signals that the resolver extracted a spurious NP, while an annotated NP is twinless when the resolver fails to extract it. Unfortunately, it is unclear how the B^3 score should be computed for twinless NPs. Bengtson and Roth (2008) simply discard twinless NPs, but this solution is likely too lenient — it doles no punishment for mistakes on twinless annotated or extracted NPs and it would be tricked, for example, by a system that extracts only the NPs about which it is most confident.

We propose two different ways to deal with twinless NPs for B^3. One option, B^3all, retains all twinless extracted NPs. It computes the precision as above when ce has a twin, and computes the precision as $1/|R_{ce}|$ if ce is twinless. (Similarly, $recall(ce) = 1/|K_{ce}|$ if ce is twinless.)

The second option, B^30, discards twinless extracted NPs, but penalizes recall by setting $recall(ce) = 0$ for all twinless annotated NPs. Thus, B^30 presumes that all twinless extracted NPs are spurious.

CEAF Score. Luo's (2005) CEAF score (for Constrained Entity-Alignment F-Measure) is a coreference resolution evaluation metric resembling the ACE score. Similar to ACE, CEAF relies on a measure of how well a response cluster matches an answer key cluster and computes an optimal mapping between answer key and response clusters. CEAF differs from ACE in that it computes recall by dividing the score of the optimal match by the score for mapping the key to itself (i.e. the maximum is 1) and precision by dividing by the score of matching the response to itself. The reported CEAF score is the F_1 score (harmonic mean of precision and recall).

Luo (2005) suggests several functions to score the goodness of the match of a key cluster A and response cluster B. We borrow one of these functions: $\phi(A, B) = (2 * |A \cap B|)/(|A| + |B|)$. In other words, the score for the match is the number of items the two clusters have in common proportional to the combined size of the two clusters.

4 Evaluation

In this section, we present experimental results to compare the performance of a coreference resolver created with Reconcile against other state-of-the-art supervised learning-based coreference resolvers.

4.1 Data Sets

For evaluation we use the six most commonly used coreference resolution data sets. Two of those are from the MUC conferences (MUC-6, 1995; MUC-7, 1997) and four are from the Automatic Content Evaluation (ACE) Program (NIST, 2004). For ACE, we use only the newswire portion because it is closest in composition to the MUC corpora. Statistics for each of the data sets are shown in Table 1. When available, we use the standard test/train split. Otherwise, we randomly split the data into a training and test set following a 70/30 ratio.

dataset	docs	CEs	chains	CEs/ch	tr/tst split
MUC6	60	4232	960	4.4	30/30 (st)
MUC7	50	4297	1081	3.9	30/20 (st)
ACE-2	159	2630	1148	2.3	130/29 (st)
ACE03	105	3106	1340	2.3	74/31
ACE04	128	3037	1332	2.3	90/38
ACE05	81	1991	775	2.6	57/24

Table 1: Dataset characteristics including the number of documents, annotated CEs, coreference chains, annotated CEs per chain (average), and number of documents in the train/test split. We use *st* to indicate a standard train/test split.

4.2 The $Reconcile_{ACL09}$ Configuration

We created a coreference resolution system using Reconcile, which we will call $Reconcile_{ACL09}$because this system was used to conduct the experiments presented in (Stoyanov et al., 2009). $Reconcile_{ACL09}$is modeled after the state-of-the art system of Ng and Cardie (2002), using many of the same features. The $Reconcile_{ACL09}$system was configured using the following subsystems and options:

1. **Preprocessing**
   (a) **Sentence Segmentation:** *OpenNLP*
   (b) **Tokenizer:** *OpenNLP*
   (c) **POS Tagger:** *OpenNLP*
   (d) **Parser:** *Berkeley parser*
   (e) **Named Entity Recognizer:** *Stanford NER system w/ additional LLNL models*
2. **Feature Set**[4] - SoonStr, ProStr, ProComp, PNStr, WordsStr, WordOverlap, Modifier, PNSubstr, WordsSubstr, Pronoun1, Pronoun2, Definite1, Definite2, Demonstrative2, Embedded1, Embedded2, InQuote1, InQuote2, BothProperNouns, BothEmbedded, BothInQuotes, BothPronouns, BothSubjects, Subject1, Subject2, Appositive, MaximalNP, Animacy, Gender, Number, SentNum, ParNum, Alias, IAntes, Span, Binding, Contraindices, Syntax, ClosestComp, Indefinite, Indefinite1, Prednom, Pronoun, ContainsPN, Constraints, ProperNoun, Agreement, ProperName, WordNetClass, WordNetDist, WordNetSense, Subclass, RuleResolve, ProResolve, SameSentence, ConsecutiveSentences, AlwaysCompatible, SameParagraph, HeadMatch, PairType, Quantity, WNSynonyms
3. **Classifier** - *Perceptron learning algorithm* - from the Weka toolkit (Witten and Frank, 2005). Run for 15 iterations.
4. **Clustering** - *Single-link Clustering* - The clustering thresholds were tuned by cross validation of the training set as described in Section 4.

4.3 Experimental Results

Table 2, box 1 shows the performance of $Reconcile_{ACL09}$ using a default (0.5) threshold for the coreference classifier. All of these results are F_1 scores (recall and precision are equally weighted). The MUC score is highest for the MUC6 data set, while the four ACE data sets show much higher B^3 scores as compared to the two MUC data sets. The latter occurs because the ACE data sets include singletons.

The classification threshold, however, can be gainfully employed to control the trade-off between precision and recall. This has not traditionally been done in learning-based coreference resolution research — possibly because there is not much training data available to sacrifice as a validation set. Nonetheless, we hypothesized that estimating a threshold *from just the training data*

[4]Full descriptions of the features are found in the Appendix.

		MUC6	MUC7	ACE-2	ACE03	ACE04	ACE05
1. 0.5 THRESHOLD	MUC	70.40	58.20	65.76	66.73	56.75	64.30
	B^3all	69.91	62.88	77.25	77.56	73.03	72.82
	B^30	68.55	62.80	76.59	77.27	72.99	72.43
2. THRESHOLD ESTIMATION	MUC	68.50	62.80	65.99	67.87	62.03	67.41
	B^3all	70.88	65.86	78.29	79.39	76.50	73.71
	B^30	68.43	64.57	76.63	77.88	75.41	72.47
3. OPTIMAL THRESHOLD	MUC	71.20	62.90	66.83	68.35	62.11	67.41
	B^3all	72.31	66.52	78.50	79.41	76.53	74.25
	B^30	69.49	64.64	76.83	78.27	75.51	72.94

Table 2: Reconcile's F_1 scores on six standard data sets.

System		MUC6	MUC7
Soon et al. (2001)	MUC	62.6	60.4
Ng and Cardie (2002)	MUC	70.4	63.4
Yang et al. (2003)	MUC	71.3	60.2

Table 3: F_1 scores for state-of-the-art coreference systems using comparable evaluation settings.

might be effective. Our results (THRESHOLD ESTIMATION box in Table 2) indicate that this indeed works well.[5] With the exception of MUC6, results on all data sets and for all scoring algorithms improve; moreover, the scores approach those for runs using an optimal threshold (box 3) for the experiment as determined by using the **test set**.

One of the goals of Reconcile was to provide a platform that can be used to build to assemble a coreference resolution system that exhibits performance that is comparable to the state-of-the-art. Unfortunately, due to the difficulties outlined in Section 1, results presented here can be compared only to a limited number of scores reported in the literature. Table 3 lists some of the best scores that are directly comparable to our system. Based on these comparisons, $Reconcile_{ACL09}$does indeeed exhibit performance levels that are comparable to the previously reported scores for state-of-the-art supervised learning-based coreference resolvers.

5 Conclusions

Reconcile was designed to be a general architecture for coreference resolution that can be used to easily create learning-based coreference resolvers. Our goal in building Reconcile was to support experimental research and to provide a state-of-the-art coreference resolver for both NLP researchers and application developers. We hope that this software infrastructure will help to advance the state-of-the-art in conference resolution by enabling researchers to easily generate experimental results that are directly comparable, yielding better insights into which ideas and techniques truly perform the best. We also hope that Reconcile will allow NLP researchers and practitioners to easily incorporate coreference resolution into larger systems for higher-level end applications.

6 Acknowledgments

This research was supported in part by Lawrence Livermore National Laboratory subcontract B573245 and the Department of Homeland Security under ONR Grant N0014-07-1-0152.

[5] All experiments sample uniformly from 1000 threshold values.

Appendix: Feature Set

This appendix contains a breif descriptions of the features currently available in Reconcile. Most of the categorical features are binary valued with two values C (for compatible) and I (for incompatible). Some features add a third value, NA for not available. The following describes the values of the features currently available in Reconcile. All features are defined over a pair of NPs – np1 and np2.

1. Agreement - The value of this feature is 'C' if the NPs agree on both number and gender. If they disagree on either number or gender, the value of the feature is 'I'. The value of the feature is 'NA' if no gender or number information is present for one or both of the NPs in question.

2. Alias - 'C' if one NP is an alias of the other, otherwise 'I'. Alias can mean a variety of things, such as different was of representing the same date, monetary value or number. This featue uses information about the semantic type of the NPs.

3. AlwaysCompatible - Always returns 'C'.

4. AlwaysIncompatible - Always returns 'I'.

5. AnaMed - This numerical feature returns: $m - \frac{med(np_1,np_2))}{m}$, where $m = len(np_1)$ and med is the minimum edit distance.

6. Animacy - If the NPs agree on animacy, returns 'C', else returns 'I'. Uses semantic type information.

7. AnteMed - Nearly the same as AnaMed, but focuses on the antecedent. Returns $m - med(np_1, np_2)/m$ where $m = len(np_2)$.

8. Appositive - Return 'C' if the NPs are in an appositive construction, else returns 'I'. Uses semantic type information.

9. Binding - Returns 'C' if the NPs do not violate conditions B and C in Chomsky's binding theory, else 'I'. In short, element α binds elements β if and only if α c-commands β. A node X c-commands node Y if and only if:

   - X does not dominate Y
   - Y does not dominate X
   - The first branching node dominating X also dominates Y.

10. BothDefinites - 'C' if both NPs are definite, 'NA' if only one contains the article, 'I' otherwise.

11. BothEmbedded - Returns 'C' if both NPs are embedded, 'NA' if only one is, and 'I' if neither is embedded.

12. BothInQuotes - Returns 'C' if both NPs are inside of quotes, 'NA' if only one is, and 'I' if neither is in quote.

13. BothPronouns - Returns 'C' if both NPs are pronouns, 'NA' if only one is, and 'I' if neither is a pronoun.

14. BothProperNames - Returns 'C' if both NPs are proper names, 'NA' if only one is, and 'I' if neither is a proper name.

15. BothProperNouns - Returns 'C' if both NPs are proper nouns, 'NA' if only one is, and 'I' if neither is a proper noun.

16. BothSubjects - Returns 'C' if both NPs are in the subject position relative to a verb clause, 'NA' if only one is, and 'I' if neither is a subject.

17. ClosestComp - This feature checks the semantic compatibility between the two NPs. If they are compatible and the closest, it returns 'C', else 'I'. Uses semantic type information.

18. ConsecutiveSentences - Returns 'C' if the NPs are in consecutive sentences, otherwise returns 'I'.

19. Constraints - Checks the compatibility of the GENDER, NUMBER, CONTRAINDICES, ANIMACY, PRONOUN and CONTAINSPN, if all are compatible (or at least not Incompatible types for the last four features), then it returns 'C', else 'I'.

20. ContainsPN - This features checks that both NPs contain proper names and contain no words in common, if this is true, it returns 'I', else 'C'.

21. Contraindices - The following constraints are implemented for this feature: (1) two NP's

separated by a preposition cannot be coindexed and (2) two non-pronominal NP's separated by a non-copular verb cannot be coindexed. If the two NP's violate these conditions, then returns 'I', else it is 'C'.

22. Definite1 - Returns 'Y' if the first NP starts with 'the', else 'N'.

23. Definite2 - Returns 'Y' if the second NP starts with 'the', else 'N'.

24. Demonstrative2 - Returns 'Y' if the second NP starts with a demonstrative, i.e. this, that, these and those. Returns 'N' otherwise.

25. DocNo - A bookkeeping (uninormative) feature – the internal number of the document from which the NPs came from.

26. Embedded1 - 'Y' if the first NP is an embedded or nested NP, else 'N'.

27. Embedded2 - 'Y' if the second NP is an embedded or nested NP, else 'N'.

28. Gender - If the two NPs agree in gender, then this feature returns 'C', and 'I' if they disagree. If the gender information is not determined by the system, then it returns 'NA'.

29. GramRole1 - The grammatical role of the first NP.

30. GramRole2 - The grammatical role of the second NP.

31. HeadMatch - Checks for matching head noun between the two NPs, if they match, returns 'C', else 'I'.

32. IAntes - Returns 'Y' if one of the two NPs is the pronoun "I" and the other NP is determined to be the quoted speaker of the text containing the "I" pronoun by a rule-based system.

33. ID1 - The identification number of the first NP – another bookkeeping feature and is not used in training nor testing.

34. ID2 - The identificaiton number of the second NP.

35. Indefinite - 'I' if the second NP is an indefinite and is not an appositive, 'C' otherwise.

36. InQuote1 - Returns 'Y' if the first NP is part of a quoted string.

37. InQuote2 - Returns 'Y' if the second NP is part of a quoted string.

38. instClass - The class label for the pair used in training – '+' if the pair is coreferent, '-' otherwise.

39. MaximalNP - If both NPs have the same maximal NP projection, then return 'I', else returns 'C'.

40. Modifier - If the prenominal modifiers of one np are a subset of the prenominal modifiers of the other nps, then returns 'C', else 'I'.

41. Number - If the two NPs agree in number, then this feature returns 'C', and 'I' if they disagree. If the number information of one or more of the NPs cannot be determined, the value is 'NA'.

42. PairType - Encodes the type of the np pair, i.e., if the pair are Proper nouns, pronouns, definite or indefinite.

43. ParNum - The distance between the two NPs as they occur in the text in terms of paragraphs.

44. PNStr -If both NPs are proper names and the same string, then 'C', else 'I'.

45. PNSubstr - If both NPs are proper names and one is a substring of the other, then 'C', else 'I'.

46. Prednom - If the NPs form a predicate nominal construction, then 'C', else 'I'. An example: "Barack Obama is the U.S. president.", *the U.S. President* is acting as the predicate nominal.

47. ProComp - If both NPs are pronouns and are compatible in gender, number and person, (i.e., *he* and *his*), then 'C', otherwise 'I'.

48. ProEquiv - If both NPs are pronouns and agree in GENDER and NUMBER and appear in consecutive sentences, then 'I', else 'C'.

49. Pronoun1 - If NP1 is a pronoun, return 'Y', else 'N'.

50. Pronoun2 - If NP2 is a pronoun, return 'Y', else 'N'.

51. Pronoun - If NP1 is a pronoun and NP2 is not, then return 'I', else 'C'.

52. ProperName - If both NPs are proper names and share no words in common, then returns 'I', else 'C'.

53. ProperNoun - If both NPs are proper nouns and share no words in common, then returns 'I', else 'C'.

54. ProResolve - If one NP is a pronoun and the other NP is its antecedent according to a rule-based algorithm, then 'C', else 'I'.

55. ProStr - Return 'C' if both NPs are pronouns and their strings match exactly, otherwise 'I'.

56. Quantity - Returns 'C' if the two NPs form the pattern ¡sum¿ of ¡money¿ (e.g. loss of 1 million), else 'I'.

57. RuleResolve - If the two NPs are coreferent according to a rule-based algorithm, then 'C', else 'I'.

58. SameParagraph - The NPs are found in the same paragraph, then return 'Y', else 'N'.

59. SameSentence - The NPs are found in the same setence, returns 'Y', else 'N'.

60. SentNum - The distance between the two NPs in terms of sentences.

61. SoonStr - If after discarding uninformative words, the strings two NPs match, then return 'C', else 'I'.

62. SoonStrNonPro - If both NPs are non-pronominal and after discarding determiners the two NPs match, then return 'C', else 'I'.

63. Span - Returns 'I' if one NP spans the other, else 'C'.

64. Subclass - If one NP's WordNet class is a subclass of the other NP return 'Y' else 'N'.

65. Subject1 - Returns 'Y' if NP1 is a subject, otherwise 'N'.

66. Subject2 - Returns 'Y' if NP2 is a subject, otherwise 'N'.

67. Syntax - If the two NP's have incompatible values for BINDING, CONTRAINDICES, SPAN, or MAXIMALNP, then returns 'I', else 'C'.

68. Title - If one or both NPs is a title, returns 'I', else 'C'.

69. WeAntes - Returns 'Y' if one of the two NPs is a form of the pronoun "we" and the other NP is determined to be the organization of the quoted speaker of the text containing the pronoun by a rule-based system.

70. WNSynonyms - Returns 'C' if the NPs are WordNet synonyms, else 'I'.

71. WordNetClass - Returns 'C' if both NPs have the same WordNet class, else 'I'.

72. WordNetDist - The distance in the WordNet Synset tree between the two NPs.

73. WordNetSense - Returns the first WordNet sense that both NPs share.

74. WordOverlap - If the intersection of the content words of the two nps is not empty, then 'C', else 'I'.

75. WordsStr - If both NPs are non-pronominal and the strings match, then 'C', else 'I'.

76. WordsSubstr - If both NPs are non-pronominal and one np is proper substrings of the other with respect to content words, then returns 'C', else 'I'.

References

A. Bagga and B. Baldwin. 1998. Algorithms for scoring coreference chains. In *In Linguistic Coreference Workshop at LREC 1998*.

Baldridge, J. 2005. *The OpenNLP project. http://opennlp.sourceforge.net/.* The OpenNLP Project.

Eric Bengtson and Dan Roth. 2008. Understanding the value of features for coreference resolution. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 294–303. Association for Computational Linguistics.

Chih-Chung Chang and Chih-Jen Lin. 2001. Libsvm: a library for support vector machines.

Cognitive Computation Group. 2009. Sentence Segmentation Tool. http://l2r.cs.uiuc.edu/ cogcomp/atool.php?tkey=SS.

J. Finkel, S. Dingare, H. Nguyen, M. Nissim, and C. Manning. 2004. Exploiting Context for Biomedical Entity Recognition: From Syntax to the Web. In *Joint Workshop on Natural Language Processing in Biomedicine and its Applications at COLING 2004*.

T. Finley and T. Joachims. 2005. Supervised clustering with support vector machines. In *International Conference on Machine Learning (ICML)*, pages 217–224.

A. Haghighi and D. Klein. 2007. Unsupervised Coreference Resolution in a Nonparametric Bayesian Model. In *Proceedings of the 45rd Annual Meeting of the Association for Computational Linguistics*.

T. Joachims. 2002. SVM light, http://svmlight.joachims.org.

D. Klein and C. Manning. 2003. Fast Exact Inference with a Factored Model for Natural Language Parsing. In *Advances in Neural Information Processing Systems 15*.

S. Lappin and H. Leass. 1994. An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4):535–561.

Xiaoqiang Luo, Abe Ittycheriah, Hongyan Jing, Nanda Kambhatla, and Salim Roukos. 2004. A mention-synchronous coreference resolution algorithm based on the bell tree. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*.

X. Luo. 2005. On Coreference Resolution Performance Metrics. In *Proceedings of the 2005 Human Language Technology Conference / Conference on Empirical Methods in Natural Language Processing*.

A. McCallum and B. Wellner. 2004. Conditional Models of Identity Uncertainty with Application to Noun Coreference. In *18th Annual Conference on Neural Information Processing Systems*.

Thomas Morton. 1999. Using coreference for question answering. In *The 8th Text REtrieval Conference (TREC-8)*, pages 85–89.

MUC-6. 1995. Coreference Task Definition. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, pages 335–344.

MUC-7. 1997. Coreference Task Definition. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*.

V. Ng and C. Cardie. 2002. Improving Machine Learning Approaches to Coreference Resolution. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*.

V. Ng. 2008. Unsupervised Models for Coreference Resolution. In *Proceedings of 2008 the Conference on Empirical Methods in Natural Language Processing (EMNLP-2008)*.

NIST. 2004. *The ACE Evaluation Plan*. NIST.

S. Petrov and D. Klein. 2007. Improved Inference for Unlexicalized Parsing. In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL 2007)*.

M. Poesio and M. Kabadjov. 2004. A general-purpose, off-the-shelf anaphora resolution module: implementation and preliminary evaluation. In *LREC*.

L. Qiu, M.-Y. Kan, and T.-S. Chua. 2004. A public reference implementation of the rap anaphora resolution algorithm. In *LREC*.

W. Soon, H. Ng, and D. Lim. 2001. A Machine Learning Approach to Coreference of Noun Phrases. *Computational Linguistics*, 27(4):521–541.

J. Steinberger, M. Poesio, M. Kabadjov, and K. Jezek. 2007. Two uses of anaphora resolution in summarization. *Information Processing and Management, Special Issue on Summarization*, 43:16631680.

V. Stoyanov, N. Gilbert, C. Cardie, and E. Riloff. 2009. Conundrums in noun phrase coreference resolution: Making sense of the state-of-the-art. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*.

Y. Versley, S.P. Ponzetto, M. Poesio, V. Eidelman, J. Jern, A.and Smith, X. Yang, and A. Moschitti. 2008. BART: A modular toolkit for coreference resolution. In *LREC*.

M. Vilain, J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman. 1995. A Model-Theoretic Coreference Scoring Theme. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*.

I. Witten and E. Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition*. Morgan Kaufmann.

Xiaofeng Yang, Guodong Zhou, Jian Su, and Chew Lim Tan. 2003. Coreference resolution using competition learning approach. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 176–183.