Easy-first Coreference Resolution

Veselin Stoyanov[1] *Jason Eisner*[1,2]
(1) HLT-COE, Johns Hopkins University
(2) CLSP, Johns Hopkins University
{ves,jason}@cs.jhu.edu

Abstract

We describe an approach to coreference resolution that relies on the intuition that easy decisions should be made early, while harder decisions should be left for later when more information is available. We are inspired by the recent success of the rule-based system of Raghunathan et al. (2010), which relies on the same intuition. Our system, however, automatically learns from training data what constitutes an easy decision. Thus, we can utilize more features, learn more precise weights, and adapt to any dataset for which training data is available. Experiments show that our system outperforms recent state-of-the-art coreference systems including Raghunathan et al.'s system as well as a competitive baseline that uses a pairwise classifier.

KEYWORDS: coreference resolution, discourse processing, supervised clustering, greedy approaches.

KEYWORDS IN L_2: resolución de correferencia.

1 Introduction

Coreference resolution has traditionally benefited from machine learning approaches (Soon et al., 2001; Ng and Cardie, 2002; Bengtson and Roth, 2008; Stoyanov et al., 2009). Surprisingly, however, recent work has shown that simple rule-based coreference systems can compete with the state-of-the-art machine-learning-based systems if provided with rich lexical, syntactic, semantic and discourse information (Haghighi and Klein, 2010; Raghunathan et al., 2010). In fact, the rule-based system of Raghunathan et al. (2010) exhibited the top score in the recent CoNLL evaluation (Pradhan et al., 2011). This system's innovation is to build the coreference clusters incrementally, starting with the most precise rules (dubbed *sieves* by Raghunathan et al. (2010)) and use the available coreference information to guide the less precise sieves.

Coreference resolution is inherently a global task: for example, discovering that *Smith* and *she* corefer makes it more probable that *Smith* corefers with *Jane Smith* (i.e., a female name) rather than *Jason Smith*. Furthermore, grouping *Jane Smith*, *she* and *Jason Smith* in the same cluster should be rejected by the clustering algorithm because it results in poor gender and proper name agreement. Yet incorporating such structured information in coreference has proven challenging. There have been several successful attempts to incorporate structured information through joint inference such as Culotta et al. (2007) and Poon and Domingos (2008), but they do not explicitly learn parameters tuned to the inference algorithm used at test time (and the latter relies on a complicated and expensive inference procedures). The difficulty of inferring globally consistent clusterings lies in the fact that there are exponentially many clusterings and producing an "optimal" clustering according to most measures of global coherence is NP-hard. In this context, the approach of Raghunathan et al. (2010) can be seen as a rule-based greedy search for a globally consistent clustering.

We propose a coreference resolution approach that like Raghunathan et al. (2010) aims to consider global consistency while performing fast and deterministic greedy search. Similar to Raghunathan et al. (2010), our algorithm operates by making the easy (most confident) decisions first. It builds up coreference clusters as it goes and uses the information from these clusters in the form of features to make later decisions. However, while Raghunathan et al. (2010) use hand-written rules for their system, we learn feature weights from training data.

What do the learned weights *mean*? They tell our system how to make the merging decisions as it performs greedy agglomerative clustering. And how do we *learn* the weights? Inspired by Goldberg and Elhadad's (2010) approach to easy-first dependency parsing, we utilize a learning method that performs supervised perceptron-style updates as it carries out clustering. Thus, during training, the learner observes partially completed clusterings similar to those that are likely to be encountered during testing.

Our approach inherits all of the advantages of discriminatively trained systems: tuning of parameters based on the empirical properties of training data (instead of hand-tuning weights); an ability to easily adapt to different datasets for which training data is available; an ability to utilize arbitrary overlapping features of the data. In addition, like Raghunathan et al. (2010), we are able to utilize information from earlier, more certain steps when making later, less certain decisions. Our experimental section compares both to traditional ML-based approaches that do not utilize incremental information and to the approach of Raghunathan et al. (2010), which is incremental but utilizes no ML. Under the same evaluation settings, our algorithm outperforms a competitive machine-learning baseline that uses a pairwise classifier, under 9 out of the 10 evaluation settings. Additionally, our system outperforms that of Raghunathan et al.

by more than 2 points on all 4 evaluation settings in which we are able to compare directly.

2 Coreference Resolution

Noun phrase coreference resolution is the task of determining whether two noun phrases (NPs) refer to the same real-world entity or concept. In this paper we will be concerned with determining coreference only within a document. Following established terminology, we use the term **mention** to refer to a linguistic expression that may participate in the coreference relation, as defined for the particular task. Typically mentions are noun phrases, but definitions vary by dataset and may include nested nouns, gerunds, etc. (see Stoyanov et al. (2009) for details). Our method is agnostic to the particular definition of what constitutes a mention, provided that it can rely on a component that extracts mentions with reasonable accuracy. We will use the terms **coreference chain** (or just **chain**) and **cluster** interchangeably to mean a set of mentions that have been posited to refer to the same entity.

The field of coreference resolution has been dominated by the mention-pair model (Soon et al., 2001; Ng and Cardie, 2002; Bengtson and Roth, 2008; Stoyanov et al., 2009). This approach trains a classifier to decide whether a pair of distinct mentions is coreferent or not. The quadratically many decisions about all pairs are then reconciled using a clustering algorithm to form the predicted coreference chains. Different features, learners, and clustering algorithms have been employed in the literature. Surprisingly, the mention-pair model with a simple clustering algorithm such as single-link clustering (i.e., transitive closure) performs on par with state-of-the-art systems (Bengtson and Roth, 2008; Stoyanov et al., 2009).

The pitfall of the mention-pair model and other algorithms that rely only on "local" information between pairs of mentions is that they cannot consolidate structured information. In fact, it is common in practice that some chains produced by such algorithms exhibit low coherency. (E.g., a cluster may consist of [*Jason Smith*, *Smith*, *Smith*, *she*], which looks good to the clustering algorithm because 5 of its 6 pairs are plausible. See section 4.4 for other examples.) In addition to negatively impacting evaluation scores, such clusters are especially irritating to human users of the system output.[1] Recent work has attempted to overcome the limitation of local models. Culotta et al. (2007) and Poon and Domingos (2008) perform global clustering of all mentions in a document by using first-order probabilistic models in the supervised and unsupervised settings respectively. However, these models do not specifically tune their weights to their respective test-time inference procedures – the method of Poon and Domingos (2008) is unsupervised, while Culotta et al. (2007) learn a scoring function that can judge the goodness of an overall clustering, but it is not trained to judge goodness incrementally as the algorithm progresses.

The model of Raghunathan et al. (2010) uses an approach based on *multiple sieves* of decreasing precision and increasing recall. It begins by creating a coreference chain for each mention in the document. Each sieve consists of deterministic tests that are applied to pairs of chains in the current clustering. When the tests succeed, the coreference chains in the pair are joined together. The sieves are manually designed and tuned manually on development data.

Another approach has considered a set of hand crafted rules — the multi-agent method of Zhou and Su (2004). Like Raghunathan et al., it relies on multiple agents that filter potential antecedent candidates. However, the system of Raghunathan et al. differs in that coreference

[1] Based on our personal experience deploying coreference resolution systems.

decisions are not made sequentially but in order of the precision of the corresponding sieves, with later (less precise) sieves relying on the information from earlier sieves.

3 Easy-first Coreference

The intuition behind the multi-pass sieve coreference system is that some decisions are easier than others. For instance, the first sieve joins together mentions that constitute the exact same string, while the second sieve looks for high-precision constructs such as appositives and predicate-nominal relations. The later sieves that make harder decisions (for instance, pronoun resolution is done by the last sieve) are joining larger clusters and so can exploit more complete information about the entities being referred to. We rely on the same intuition, but we allow the system to automatically learn what constitutes easy decisions based on features of the clusters that are to be joined. We hope that the system will learn based on the statistics of the training data, in addition to the human intuition that goes into designing the features. For example, we hope that our algorithm can learn to link *Smith* and *she* early in certain contexts (e.g., if *Smith* is the only possible antecedent for the pronoun *she*). Additionally, in contrast to a system for which rules are designed manually, our approach can utilize a large number of features, using the learned weights to aggregate evidence from different features.

We will describe our algorithm by first discussing the test-time system and then the learning algorithm that we use. We face a reinforcement learning setting. Our agglomerative clustering "agent" observes a current *state*, which consists of all current partially formed coreference chains. In the *start state*, each mention is a separate, single-element chain. At each step, the agent will select some action of the form JOIN_{ij}, which joins existing chains i and j (changing the state and the set of actions available at the next time step). Alternatively , the agent may select the special action HALT, which stops the computation and returns the current coreference chains.

3.1 Test-time Inference

At test time our agent operates greedily (without lookahead), choosing its action at each step according to a linear model. The main loop of the test-time algorithm without optimizations discussed below is shown in Algorithm 1. For each action JOIN_{ij} that is available in the current state C, we compute a feature vector $\phi(\text{JOIN}_{ij}, C)$. In this work, ϕ ignores most of C and extracts only features that depend only on the two clusters i and j. However, in the future work section we discuss features that express the relative confidence of linking clusters i and j as *compared to* the alternatives (i.e., is i the only reasonable antecedent for cluster j?). Similarly, $\phi(\text{HALT}, C)$ defines a feature vector for the HALT action. At present, there is a single dedicated feature that always fires on this action, but in future work we can use additional features that consider whether it is a good idea to HALT in the current state or to continue merging clusters.

The score of an action a in state s is given by a linear combination of the features, $w \cdot \phi(a,s)$, where w is a weight vector (coefficient) that specifies a weight for each feature. We learn w using a perceptron-style procedure (described in the next section).

Given the feature weights, inference is easy: we keep picking the action a with the highest score. If the action is HALT, we stop and return the current clustering. Otherwise, the action has the form JOIN_{ij} and we merge the clusters i and j. For the sake of efficiency, we maintain a priority queue containing all currently available actions and their current scores. At a state that has n clusters, the priority queue contains $O(n^2)$ actions.[2] When we pop JOIN_{ij} and merge i

[2] In Section 3.2 we describe a method for limiting the number of actions that we consider, so the true number of

Input: document d; weight vector w
Output: clustering C
$C =$ initial clustering with each mention in d in its own cluster ;
$A =$ empty priority queue ;
A.insert(HALT, $w \cdot \phi(\text{HALT}, C)$) ;
foreach *pair of mentions* i, j **in** d **do**
  A.insert(JOIN_{ij}, $w \cdot \phi(\text{JOIN}_{ij}, C)$) ;
end
repeat
  $a_{max} = A$.popMax() ;
  if a_{max} *has the form* JOIN_{ij} **then**
    performJoin(i, j, C, A) ; /* modifies C and A as described in text */
until $a_{max} ==$ HALT ;
return C ;

Algorithm 1: *Inference method: "easy-first" agglomerative clustering.*

and j, we update the queue accordingly, which involves removing the $O(n)$ actions that use the old i and j, and adding $O(n)$ new actions involving the new cluster ij. To quickly identify the actions to remove, each cluster stores pointers to the enqueued actions that involve it.

Additionally, each cluster stores a best-guess description of the *properties* of the entity to which the cluster refers. These properties include the *gender*, *animacy* and *number* of the entity, its *semantic type* (i.e., *person*, *organization*, etc.) as well as the set of *proper names* used to refer to the entity. When joining two clusters we consolidate the property values, which may be in conflict (i.e., the *number* of cluster i is *single*, while the *the number* of cluster j is *plural*). In principle, the cluster should store a probability distribution over the values of its properties, using this distribution both to score the compatibility with other clusters and to compute a consensus distribution when clusters are merged. Our clusters instead use a simpler approach of storing a single value for each property together with a confidence in that value. We employ manually assigned confidences consisting of three tiers. Pronouns induce the most confident properties (confidence=3), followed by proper names (confidence=2) and common nouns (confidence=1). When two clusters are merged, we resolve conflicting values of a property by choosing the more confident value, or in the case a tie, the value contributed by the larger cluster. In case of a further tie, we use the value of the cluster whose first mention is earlier in the document.

If n is the number of mentions in the document, then there are $\leq n$ clusters at any time. Furthermore the algorithm must HALT after at most n JOIN steps—and usually halts much sooner since it would be incorrect to merge all mentions into a single large cluster. Each step of the algorithm involves:

- popping the highest-scoring action JOIN_{ij} from the priority queue (runtime $O(\log n)$),
- performing the merge action (runtime $O(1)$ by having the new cluster ij point back to the old clusters i and j),
- computing the properties of the newly merged cluster (total runtime $O(1)$),
- deleting $O(n)$ old actions (total runtime $O(n \log n)$),
- computing the scores of $O(n)$ new actions (runtime $O(n)$),[3]
- and inserting these (runtime $O(n \log n)$ or $O(n)$ depending on the type of priority queue).

actions is $O(Cn)$, where C is a constant.

[3]One might think that computing the score of each new JOIN_{ij} action would be expensive (because one might have

Thus, the total runtime is $O(n^2 \log n) + O(sn \log n)$, where $s \leq n$ is the number of actions popped before HALT and tends to be small. The initial $O(n^2 \log n)$ term is for initializing the priority queue by pushing all actions JOIN_{ij}. We will speed this up below by pruning the set of actions.

Note that this easy-first algorithm essentially has the same structure as Kruskal's (1956) minimum spanning tree algorithm on a complete graph of $O(n^2)$ edges, except that as the connected components grow and merge, the weights of the edges between them are recomputed. In addition, the easy-first algorithm may HALT early, outputting a forest of several components rather than a single spanning tree. Many agglomerative clustering algorithms have this structure. The key difference here is that we are going to *learn* a linear function that weights the old and new edges—we will learn to define "easiness" such that the easy-first algorithm will achieve good scores on a coreference task.

3.2 Pair Selection

In the name of efficiency, we apply several rules to limit the number of pairs of mentions that our algorithm considers. (That is, we run our Kruskal's-like algorithm on a sparse graph rather than a complete graph with $O(n^2)$ edges.) For each mention, we only consider some of the preceding mentions as possible antecedents. The number of preceding antecedents we consider is based on the type of each mention and relies on linguistic intuitions. The following describes the possible antecedents that we consider for each mention type.[4]

- **Proper Name (Named Entity):** A proper name is compared against all proper names in the current sentence and the 40 preceding sentences. In addition, it is compared to all other mentions in the 3 preceding sentences.
- **Definite noun phrase:** Compared to all mentions in the 4 preceding sentences.
- **Common noun phrase:** Compared to all mentions in the 2 preceding sentences.
- **Pronoun:** Compared to all mentions in the three preceding sentences unless a first person pronoun. First person pronouns are additionally compared to first person pronouns in the preceding 40 sentences.

These simple rules cover almost all positive pairwise links in our corpora while effectively reducing the runtime of our algorithm from $O(n^2 \log n)$ to $O(Cn \log n)$. Technically, C as defined above is not a constant since the k preceding sentences could be arbitrarily long and thus could contain up to $n-1$ mentions. So we set a limit of at most 500 preceding mentions that we consider. This limit is never achieved in our experiments.

3.3 Features

As mentioned before, an advantage of our approach over the pairwise model is that it is not "edge-factored." Each greedy decision can rely on all of the information contained in the coreference chains built so far. In particular, we can rely on features that attempt to capture agreement between entire clusters. We will refer to such features as *cluster features*. Our cluster features include features that fire when the two clusters have the *same* gender, animacy, number,

to score all pairs of mentions in the clusters being joined). However, we explain in the next section how to do it in $O(1)$ time for the particular set of features that we use.

[4] The numbers that appear in our pair selection heuristics could, of course, be automatically tuned on development data to achieve a user's desired balance between speed and accuracy on a particular domain. We do not do this in the present paper, which focuses on a different kind of automatic tuning.

head noun or semantic class. Similarly, we have features that fire when the two clusters have the *same* or *compatible* gender, animacy, number, head noun or semantic class (i.e., one cluster has *gender* value *feminine* while the other has value *unknown*). We also have features that are *true* when the set of heads of the two clusters overlap or the set of proper names overlap. Another highly effective feature that we use detects when there is a conflict between person names—for instance, this feature will fire if we try to match *John Smith* and *Jane Smith*.

Additionally, each cluster stores a bit that indicates whether the cluster's first mention is a pronoun (where "first" refers to the order in which the mentions appear in the text). If it is, additional features indicate whether the other cluster is a potential antecedent for the pronoun (i.e., whether it agrees in gender, number, and/or animacy and has a non-conflicting semantic type and has at least one mention that appears earlier). Additional features indicate whether the noun phrase is the only such potential antecedent in the sentence in which the pronoun appears; the only potential antecedent in the preceding sentence; or the only potential antecedent that is a verbal subject in the preceding three sentences. This special handling is necessary to assure pronouns to get a single antecedent.

In addition to the cluster features, our system utilizes the rich set of features developed for mention-pair systems that capture local configurations indicative of coreference. These *local features* operate on pairs of mentions. We include features that capture when the two mentions are in an appositive relation, a predicate-nominal relation or are aliases of each other (see Ng and Cardie (2002) and Raghunathan et al. (2010) for descriptions of these features). We also introduce in the local feature set some features that capture the textual overlap of two mentions. We use most of the local features utilized by Stoyanov et al. (2009), with the exception of the ones that duplicate our cluster features. Details about the local features that we use can be found in Stoyanov et al. (2009) and Raghunathan et al. (2010). As discussed later in the paper, the code for our system is available upon request; it documents both the feature names as they relate to the corresponding papers, as well as their semantics.

Again, each local feature evaluates a *pair of mentions*. However, what we are evaluating is a JOIN_{ij} action that merges a *pair of clusters*. Thus, when we consider joining two clusters i and j at least one of which contains more than one mention, we have a choice of all local vectors between mentions in i and j to score the local compatibility. After some experimentation on development data, we decided to select the local cluster vector which results in the highest overall local score. In other words, we pick the single local vector that scores highest under w. This resembles single-link clustering. For example, to create a long chain of mentions that runs through the text, it would suffice for each pair of *adjacent* mentions (for example) to be locally compatible (e.g., syntactically related or proximate). Even though other mention pairs in the chain might not receive a high pairwise score (because they are not *directly* related), these lower pairwise scores would not be able to veto the chain. Thus, we effectively have a division of labor where positive evidence is contributed mainly by local pairwise features and negative evidence is given only by cluster features that detect global incompatibilities.

At test time, maintaining the local feature vectors takes time $O(1)$ since both the local features and the weight vector w are fixed. Thus, we can maintain a single local feature vector for each action. When joining two clusters i and j into ij, for each k, the local feature vector of the new action $\text{JOIN}_{(ij)k}$ becomes the maximum of the local feature vectors of actions JOIN_{ik} and JOIN_{jk}. At training time, w is not fixed, so we need to maintain local vectors for all pairs of mentions and recompute the maximum every time w changes.

3.4 Learning

The goal of training is to find a weight vector w that leads our easy-first algorithm to good clusterings. Goodness is defined in terms of a task-specific evaluation function (we consider MUC score and the BCubed evaluation metrics described in Section 4). We face a typical reinforcement learning problem in that in effect, our training set depends on our weight vector w—if we choose our actions differently or choose a different input document, we end up in previously unvisited states, which offer new sets of actions to choose among. During training we need to act in the environment to generate states that present the kinds of choices among actions that we are likely to encounter during testing. We cannot practically enumerate every possible state even for a single training example, because there are exponentially many clusterings; and of course test examples will give rise to entirely new states.

In early experiments we attempted using standard reinforcement learning algorithms such as Q-learning (Watkins and Dayan, 1992) and policy gradient (Sutton et al., 2000), but the learned weights did not lead to accurate predictions. We believe that this is due to the large variation of the values of our actions. Thus, we settled on a variant of the structured perceptron with early updates (Collins and Roark, 2004) with beam width 1. Our algorithm was inspired by the successful use of a similar algorithm by Goldberg and Elhadad (2010) for training an easy-first dependency parser. The nature of our task necessitates certain differences from both the structured perceptron and the "easy-first" algorithm of Goldberg and Elhadad. Those differences are described bellow.

Algorithm 2 is our training algorithm. Overall, the algorithm monitors the agent's behavior, and performs perceptron-style updates when the current weights would result in a mistake. Learning starts from a vector of initial weights $w^{(0)}$ (initialization is discussed below) and iterates over all training documents. For each document d training begins at the start state, where each mention is in its own cluster (we use C to refer to the current set of clusters and A to refer to the current set of available actions). A state s is specified by C and A. At each time step, t, the algorithm finds the highest weighted action a_{max} according to the current weights $w^{(t)}$. If a_{max} is positive (we say that an action is positive if it results in an immediate increase of the evaluation metric for which we train), w is not updated. If a_{max} is negative (i.e., results in an immediate decrease in evaluation score), the algorithm performs a perceptron update by subtracting the features of a_{max} and adding the features of the highest scoring positive action a_{pos} scaled by a learning rate η: $w \mathrel{+}= \eta \cdot (\phi(a_{pos}, C) - \phi(a_{max}, C))$. We repeat this update k times or until a_{max} is a positive action. Note that the highest-scoring positive action a_{pos} may be different in each iteration as the feature weights are changing. Once the weights have been updated, the algorithm selects the (new) highest-scoring action a_{max} (which may be positive or negative at this point) and performs it. The procedure is repeated in the new state, and so on. When the selected action is the HALT action, we stop clustering the document and move on to the start state for the next document. After iterating over the corpus a specified number of times, the learner returns the average of all the weight vectors that were actually used to choose actions.

The above algorithm learns a weight for the HALT action. Nevertheless, in an additional step after running Algorithm 1, we fine-tune the weight of the single identity feature associated with the HALT action. We are essentially learning a *threshold* on when to stop resolution. Stopping earlier will improve precision at the expense of recall, so the ideal time to stop depends on our evaluation metric. We freeze all other feature values and pick the weight of the HALT feature

Input: set D of coreference-annotated documents; initial weight vector w; learning rate η; number of iterations *numIter*; max number of updates per error, k
Output: new weight vector w

$w_{sum} = \vec{0}$; $wcount = 0$;
for $i = 1$ **to** *numIter* **do**
  foreach *document* $d \in D$ **do**
    initialize C and A from d as in the first 5 lines of Algorithm 1 ;
    repeat
      for *tries* $= 1$ **to** k **do**
        $a_{max} = A$.peekMax() ; /* highest-scoring action */
        if a_{max}.isPositive() **then break** ; /* proceed with it if it's positive */
        $a_{pos} = A$.peekMaxPositive() ; /* highest-scoring pos action (or Halt if none) */
        $w \mathrel{+}= \eta \cdot (\phi(a_{pos}, C) - \phi(a_{max}, C))$; /* update toward a_{pos} */
        recompute scores in A using new w ;
      end
      $w_{sum} \mathrel{+}= w$; $wcount$++;
      $a_{max} = A$.popMax() ;
      if a_{max} *has the form* JOIN_{ij} **then**
        performJoin(i, j, C, A) ;
    until $a_{max} ==$ HALT ;
  end
end
return $w_{sum}/wcount$;

Algorithm 2: *A training method for easy-first clustering.*

that maximizes the desired evaluation metric on the training data. Experiments on development data showed that this step results in rather small but consistent improvements in our domain.

As noted above, our algorithm is a slight modification of the structured perceptron with early updates (Collins and Roark, 2004) for beam width 1. The difference is that upon making a mistake on a training example (in our case a training example is a document), the structured perceptron updates the weights and moves onto the next example. In contrast, we update the weights and continue working on the current example. Our subsequent updates on the example may thus have to update from one incorrect clustering to another—unlike the structured perceptron or violation-fixing perceptrons in general (Huang et al., 2012) (but like DAgger (Ross et al., 2011)). The difference is motivated by the fact that our data is highly non-separable—it is extremely rare to get all of the coreference decisions right in a document. Thus, the algorithm needs to operate in states in which some errors were previously made. Additionally, there is a significant overhead in initializing the necessary data structures for each document. Thus, a straightforward implementation of the structured perceptron with early updates is inefficient.

Goldberg and Elhadad (2010) use an update strategy similar to the structured perceptron, but keep updating until the next move will not result in an error ($k = \infty$).[5] For the reasons discussed above, such an update is not suitable for our problem—we want the algorithm to learn to operate as well as possible in states that contain errors, as such states will inevitably be encountered during testing.

[5] Because it was unclear to them at the time how to supervise the parser beyond its first error (Goldberg, p.c.).

Features from Stoyanov et al. (2009)
ProComp, SoonStr, Modifier, PostModifier, WordsSubstr, Pronoun1, Pronoun2, Definite1, Definite2, Demonstrative2, Embedded1, Embedded2, InQuote1, InQuote2, BothProperNouns, BothEmbedded, BothInQuotes, BothPronouns, BothSubjects, Subject1, Subject2, Appositive, RoleAppositive, MaximalNP, SentNum0, SentNum1, SentNum2, SentNum3, SentNum4plus, ParNum0, ParNum1, ParNum2plus, Alias, Acronym, IAntes, WeAntes, BothYou, Span, Binding, Contraindices, Syntax, ClosestComp, Indefinite, Indefinite1, Prednom, Pronoun, ProperNoun , WordNetClass, WordNetDist, WordNetSense, Subclass, AlwaysCompatible, WNSynonyms, Quantity, PairType
Features from Raghunathan et al. (2010)
IwithinI, Demonyms, ModifierHeadMatch, WhoResolve, WhichResolve
Novel features:
DeterminerHeadMatch, Longer2, LongerPN2, ShorterPN2, Halt

Table 1: Names of the local features used. Global features are described at start of section 3.3.

In the experiments described in the next section, we use for all runs $\eta = .01$ and $numIter = 5$ iterations of training and a single update to the training vector ($k = 1$). These parameters were tuned on additional development data (namely, the corpus described in Hasler et al. (2006)).

For our baseline (described in more detail in the next section) we train a "flat" classifier that implements the traditional mention-pair model. The baseline does not maintain clusters incrementally, but instead, it makes pairwise decisions up front and forms clusters by transitive closure. As in the easy-first case, we utilize a linear classifier trained using the perceptron algorithm on the set of initial clusters C. Weights are trained to recognize whether two clusters are coreferent or not. This training regime is equivalent to using easy-first training on the initial state without ever performing any action, but iterating over all possible actions and updating when a positive action would be classified as negative and vice versa. Our easy-first learning algorithm is initialized with the weights obtained from a single iteration of baseline training.

4 Experiments

We implemented our algorithm in Reconcile, a research platform for coreference resolution (Stoyanov et al., 2010b,a). We used the same set of preprocessing components as Stoyanov et al. (2009) and took a subset of their features for our local features. The names of the features that we use are listed in Table 1 and are documented in our code. The code of our system is available upon request. We can also provide a trained version of our easy-first algorithm that can be used as a resolver.

4.1 Data

We experiment with five of the most commonly used coreference resolution data sets. Those include two corpora from the MUC conferences (MUC-6, 1995; MUC-7, 1997) and three from the Automatic Content Evaluation (ACE) Program (NIST, 2004). We use the training and test splits previously used by Stoyanov et al. (2009).

We first evaluate an end-to-end coreference system that automatically discovers mentions. We measure system performance using two common scoring algorithms—MUC (Vilain et al., 1995) and BCubed (Bagga and Baldwin, 1998). When working with automatically extracted mentions, rather than gold-standard mentions, we use the BCubed variant proposed by Stoyanov et al. (2009), which they label BCubed_{all}.

We also compare to two unsupervised systems that have claimed the best scores on recent evaluations: the systems of Haghighi and Klein (2010) and Raghunathan et al. (2010). In order to compare to those, we also train and test a version of our system that uses gold-standard mentions on the MUC6 and ACE04 corpora. For training on the MUC6 corpus we follow the standard training/test split and report results on the test part. Unsupervised systems (Haghighi and Klein, 2010; Raghunathan et al., 2010) test on the entire newswire portion of the ACE04 corpus. In order to test the supervised methods on the same corpus, we randomly split it in two, and test on each half when training on only the other half (two-fold cross validation).

4.2 Baselines

We compare to a baseline that trains a pairwise classifier and then performs transitive closure. This setting is similar to the state-of-the-art systems of Bengtson and Roth (2008) and Stoyanov et al. (2009). The baseline system uses the same linear scoring function and the same feature set as our easy-first system, including global features. However, it only scores the pairwise joins of the initial single-mention clusters. It then simply performs transitive closure on the pairs whose scores are above the halting threshold, rather than computing new join scores during agglomerative clustering. As noted previously, the baseline is trained using the standard averaged perceptron (Freund and Schapire, 1999) and the positive decision threshold is tuned on training data as in the easy-first system.

Our pairwise baseline uses the same data splits and pre-processing components as Stoyanov et al. (2009). Thus, our results are directly comparable. In fact, our baseline can be considered a re-implementation of Stoyanov et al. (2009), except for the feature set. Our local features are a subset of the features used by Stoyanov et al. (2009), but we have also added some of the features described by Raghunathan et al. (2010) (as discussed in Section 4.3). We will use *Pairwise* to refer to this baseline in the results.

Our *Easy-first*$_{percep}$ setting runs our easy-first inference algorithm (Algorithm 1) but using the same perceptron-trained weights as the baseline system while still utilizing the global features. Finally, *Easy-first*$_{struct}$ also runs easy-first inference, but after training with the full algorithm described in Section 3.4 and Algorithm 2.

4.3 Results

Results of our fully automatic, end-to-end coreference resolution system are shown in Table 2. Results using gold standard mentions are shown in Table 3.

Comparison to Stoyanov et al. (2009). As previously mentioned, our *Pairwise* baseline (the second result line in Table 2) is a reimplementation of Stoyanov et al. (2009) (the first result line) with a different feature set. The results are quite different—*Pairwise* performs slightly worse than Stoyanov et al. on the MUC6 corpus and substantially worse on MUC7, while it shows substantial gains across the three ACE corpora. For instance, we gain 6 points of MUC F1-score on the ACE04 corpus. We attribute these differences to the different features and resources that we used. The main differences are outlined below.

First, *Pairwise* drops several of the features used by Stoyanov et al. (2009). Most notably, we drop the *RuleResolve* and *ProResolve* features that they utilize. Those features rely on an internal rule-based coreference and anaphora resolution systems, respectively, and indicate whether the respective rule based system clustered the noun-phrases together. We chose not to utilize these

	MUC6		MUC7		ACE03		ACE04		ACE05	
	B^3	*MUC*	B^3	*MUC*	B^3	*MUC*	B^3	*MUC*	B^3	*MUC*
Stoyanov et al.	70.9	**68.5**	**65.9**	**62.8**	79.4	67.9	76.5	62.0	73.7	67.4
Pairwise baseline	70.7	67.9	62.2	57.9	81.7	73.5	79.4	68.0	75.7	70.2
Easy-First$_{percep}$	71.5	68.4	64.4	58.1	82.8	74.1	80.3	68.4	**76.0**	70.0
Easy-First$_{struct}$	**72.1**	68.2	64.6	57.7	**83.1**	**74.5**	**80.9**	**68.8**	75.9	**70.9**

Table 2: Results for "end-to-end" coreference resolution, where mentions are automatically extracted. The best result in each column is boldfaced.

	MUC6		ACE04	
	B^3	*MUC*	B^3	*MUC*
Haghighi and Klein	75.0	81.9	76.9	76.5
Raghunathan et al.	73.2	77.7	78.9	78.1
Stoyanov et al.	76.1	**88.2***	*77.8*	*76.2*
Pairwise baseline	72.7	**88.2***	75.9	78.9
Easy-First$_{percep}$	73.3	**88.2***	78.4	**79.8**
Easy-First$_{struct}$	**77.5**	**88.2***	**81.8**	**80.1**

Table 3: Results with gold-standard mentions. In each column, the best result is boldfaced along with all statistically indistinguishable results (paired permutation test, $p < .01$). A star indicates that the result was achieved by a degenerate clustering—the algorithm learns a threshold that groups all mentions in a single cluster.[5] Italics indicate that result are not comparable to the rest, because Stoyanov et al. (2009) use a different test/train split and evaluate on only a subset of the ACE04 documents. Note also that the first two systems are unsupervised.

features because the rule-based systems are computationally expensive. They also appear to be engineered with significant amounts of knowledge and seem to be targeted toward the MUC corpora. We confirmed empirically that the decrease of performance on the MUC7 corpus is due to the absence of these features.

Second, as noted earlier, *Pairwise* introduces several new features inspired by Raghunathan et al. (2010). Below we list the most useful new features as shown through ablation studies:

- **Demonyms** – this feature is true if one of the two mentions is a demonym of the other, e.g. *Holland* and *Dutch*. We utilize the list of demonyms from Raghunathan et al. (2010).
- **WhoResolve** and **WhichResolve** – true for mentions that are animate/inanimate relative pronouns and the preceding mention in the text. This features can help resolve construct such as "*Smith*, *who* was present . . . ".
- **CountryCapital** – true if one mention is a country and the other is the capital of that country. This feature is useful since capitals are often used to refer to the country governments, e.g. "*London* issued a statement." This feature is novel to this work. We used a list of country capitals that we mined from Wikipedia.

Third, following Raghunathan et al. (2010), we use the list of name/gender associations provided by Bergsma and Lin (2006). This list is more comprehensive and more precise than the list that comes with the Reconcile system.

[5] Previous work (Stoyanov et al., 2009) has pointed out that because MUC annotates only anaphoric mentions, such a baseline results in a surprisingly strong performance when evaluating on MUC data with gold-standard mentions and using the MUC score.

Effectiveness of the Easy-First approach. Table 2 shows that easy-first inference, when trained using either our full algorithm or the perceptron algorithm, performs better than the pairwise baseline in 9 out of the 10 end-to-end evaluations. This suggests that it may also work better in other evaluations.[6] In general, the advantage of easy-first is stronger on the more precision-conscious BCubed score, which can be expected since easy-first mainly aims to correct precision errors (see section 4.4 for examples).

The improvements are more pronounced when we evaluate our system on gold-standard mentions as shown in Table 3. Compared to the unsupervised systems of Haghighi and Klein (2010) and Raghunathan et al. (2010), our system shows improvements of at least 2 points on all evaluation metrics on both datasets. When evaluated on BCubed score, easy-first outperforms the pairwise baseline by 5–6 points on both corpora, and Algorithm 2 now helps substantially.

For the gold-standard experiments, we test the statistical significance of these measured improvements, treating each test document as an independent observation. A paired permutation test ($p < .01$) reveals that improvements of Easy-first$_{struct}$ over Raghunathan et al. (2010) are statistically significant in all four settings. Additionally, Easy-first$_{struct}$ improvements over the pairwise baseline are statistically significant in the two settings that use BCubed score for evaluation. For the two settings using MUC score, the pairwise baselines as well as our methods find a degenerate solution that works well for that evaluation measure—the algorithm learns a threshold that groups all mentions in a single cluster.

To the best of our knowledge, the results presented in Table 3 represent a new state-of-the-art for systems using gold-standard mentions. For end-to-end coreference systems, our results in Table 2 represent a new state-of-the-art for the three ACE corpora.

4.4 Qualitative Evaluation

The cold hard numbers behind our evaluation tell only half of the story. Looking at the output of the end-to-end system reveals that the easy-first system produces results that look more consistent with human expectations. Consider, for instance the following sentence from an ACE05 document: *"The court order was requested by Jack Welch's attorney, Daniel K. Webb, who said Welch would likely be asked about his business dealings, his health and entries in his personal diary."*

The pairwise baseline is confused by the evidence that suggests that *his* should be linked to both *Welch* and *Daniel K. Webb* and incorrectly assigns all eight of the underlined mentions to the same cluster. Our easy-first approach, in contrast, does not attempt to join clusters that contain conflicting person names. Thus, it correctly separates {*Jack Welch's attorney, Daniel K. Webb* and *who*} into one cluster and all other mentions into another cluster.

Another example from the ACE04 corpus concerns resolving the following sentence: "Tamara Maschino, *for example,* a resident of *the Clear Lake area* of *Houston, criticizes* Bush *for* his *lack of attention to* pollution problems *from* chemical plants *near her* home."

Here the pairwise baseline resolves the pronoun *her* to *Bush*, grouping together {*George Bush* (an earlier occurrence), *Bush*, *his*, *her*}. The easy-first system, in contrast, first links the mention *Bush* to *George Bush*, and then correctly links *her* not to this male cluster but to *Tamara Maschino*.

[6] Indeed, 9 improvements out of 10 would be significant under a sign test ($p < 0.025$), if the 10 settings were independent. They are not, though, since the BCubed and MUC scores on a given corpus are presumably correlated.

5 Related Work

Related work on coreference resolution is discussed throughout the paper. As previously mentioned, our learning algorithm follows the easy-first approach to dependency parsing proposed by Goldberg and Elhadad (2010). Our learning method is also inspired by the structured perceptron and its application to incremental parsing (Collins and Roark, 2004). Our algorithm is an instance of learning parameters for a fixed approximate inference method (i.e., greedy search). Other approaches that learn for a fixed greedy inference method include SEARN (Daumé et al., 2009) and LaSO (Daumé III and Marcu, 2005). Other work proposes methods for learning parameters for fixed approximate methods beyond greedy search. Stoyanov et al. (2011) propose a back-propagation-based algorithm for learning the weights of Markov Random Fields and Conditional Random Fields for a fixed variational inference algorithm (they experiment with loopy belief propagation). Experiments on three NLP problems show that this regime of learning leads to more accurate system performance than traditional approximate maximum-likelihood training (Stoyanov and Eisner, 2012).

Concurrent to us, Jain et al. (2011) propose a method for learning features to guide a greedy agglomerative clustering algorithm. They apply their algorithm to a very different problem – image segmentation by clustering superpixels. Their learning algorithm is inspired by on-policy reinforcement learning and it differs from ours. Initial experiments with several reinforcement learning approaches on our problem (including Q-learning (Watkins and Dayan, 1992) and policy gradient (Sutton et al., 2000)) did not lead to improvements in our setting.

6 Conclusions and Future Work

We presented a novel algorithm for coreference resolution that capitalizes on the idea that early "easier" decisions can be used to guide later, "harder" decisions. We presented a training algorithm based on the structured perceptron for automatically learning feature weights. When evaluated on coreference data, our algorithm outperformed a competitive baseline as well as previously published state-of-the-art methods under most evaluation conditions.

Our model is still missing the opportunity to compare each action to alternatives. For instance, the algorithm should be able to learn that it is "easy" to join clusters i and j when i is the only reasonable antecedent cluster for j (there are no strong competitors). For instance, if j is headed by a pronoun and cluster i contains the only reasonable antecedent for J, the algorithm can have more confidence that i and j should be joined. Thus, it would learn to perform such actions sooner. Since actions can rely on any features of the current state, we could augment our model with such competition-style features, as used by Yang et al. (2003).[7]

Our easy-first approach is suitable for performing coreference resolution jointly with other tasks. We are particularly interested in jointly performing within-document coreference and cross-document coreference (or entity linking). An easy-first approach will allow us to trade-off decisions on the document and intra-document level. For example, linking *Obama* to the appropriate database entity may help us link it to the mention *president* later in the document.

Acknowledgments

Thanks to the anonymous reviewers, and to Yoav Goldberg, for their thoughtful comments on an earlier version. Also thanks to the JHU NLP group and especially Shane Bergsma for early

[7] We do already use competition-style features to evaluate pronoun antecedents, as described in section 3.3.

discussions of the approach. This work was supported by the the Human Language Technology Center of Excellence at Johns Hopkins University, and by National Science Foundation under Grant No. 0964681 to the second author.

References

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

Bengtson, E. and Roth, D. (2008). Understanding the value of features for coreference resolution. In *Proceedings of EMNLP*, pages 294–303.

Bergsma, S. and Lin, D. (2006). Bootstrapping path-based pronoun resolution. In *Proc. of ACL*.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*.

Culotta, A., Wick, M., Hall, R., and McCallum, A. (2007). First-order probabilistic models for coreference resolution. In *Proceedings of HLT/NAACL*, pages 81–88.

Daumé, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine learning*, 75(3):297–325.

Daumé III, H. and Marcu, D. (2005). Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of ICML*, pages 169–176.

Freund, Y. and Schapire, R. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

Goldberg, Y. and Elhadad, M. (2010). An efficient algorithm for easy-first non-directional dependency parsing. In *Proceedings of NAACL*, pages 742–750. Association for Computational Linguistics.

Haghighi, A. and Klein, D. (2010). Coreference resolution in a modular, entity-centered model. In *Proceedings of NAACL*, pages 385–393.

Hasler, L., Orăsan, C., and Naumann, K. (2006). NPs for events: Experiments in coreference annotation. In *Proceedings of LREC2006*.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of NAACL*.

Jain, V., Turaga, S., Briggman, K., Helmstaedter, M., Denk, W., and Seung, H. (2011). Learning to agglomerate superpixel hierarchies. In *Proceedings of NIPS*.

Kruskal, J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50.

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)*.

Ng, V. and Cardie, C. (2002). Improving machine learning approaches to coreference resolution. In *Proceedings of ACL*, pages 104–111.

NIST (2004). *The ACE Evaluation Plan*. National Institute of Standards and Technology, United States of America.

Poon, H. and Domingos, P. (2008). Joint unsupervised coreference resolution with Markov logic. In *Proceedings of EMNLP*, pages 650–659.

Pradhan, S., Ramshaw, L., Marcus, M., Palmer, M., Weischedel, R., and Xue, N. (2011). Conll-2011 shared task: Modeling unrestricted coreference in ontonotes. In *Proceedings of CoNLL 2011*, pages 1–27.

Raghunathan, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D., and Manning, C. (2010). A multi-pass sieve for coreference resolution. In *Proceedings of EMNLP*, pages 492–501.

Ross, S., Gordon, G. J., and Bagnell, J. A. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of AISTATS*.

Soon, W., Ng, H., and Lim, D. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational linguistics*, 27(4):521–544.

Stoyanov, V., Cardie, C., Gilbert, N., Riloff, E., Buttler, D., and Hysom, D. (2010a). Coreference resolution with Reconcile. In *Proceedings of the ACL Short Papers*, pages 156–161.

Stoyanov, V., Cardie, C., Gilbert, N., Riloff, E., Buttler, D., and Hysom, D. (2010b). Reconcile: A coreference resolution research platform. Technical report, Cornell University.

Stoyanov, V. and Eisner, J. (2012). Minimum-risk training of approximate CRF-based NLP systems. In *Proceedings of NAACL*.

Stoyanov, V., Gilbert, N., Cardie, C., and Riloff, E. (2009). Conundrums in noun phrase coreference resolution: Making sense of the state-of-the-art. In *Proceedings of ACL/IJCNLP*, pages 656–664.

Stoyanov, V., Ropson, A., and Eisner, J. (2011). Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proceedings of AISTATS*.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *NIPS*.

Vilain, M., Burger, J., Aberdeen, J., Connolly, D., and Hirschman, L. (1995). A model-theoretic coreference scoring theme. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*.

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Yang, X., Zhou, G., Su, J., and Tan, C. (2003). Coreference resolution using competition learning approach. In *Proceedings of ACL*, pages 176–183.

Zhou, G. and Su, J. (2004). A high-performance coreference resolution system using a constraint-based multi-agent strategy. In *Proceedings of COLING*.