Prune-and-Score: Learning for Greedy Coreference Resolution

Chao Ma, Janardhan Rao Doppa†, J. Walker Orr, Prashanth Mannem
Xiaoli Fern, Tom Dietterich and **Prasad Tadepalli**
School of Electrical Engineering and Computer Science, Oregon State University
{machao,orr,mannemp,xfern,tgd,tadepall}@eecs.oregonstate.edu
† School of Electrical Engineering and Computer Science, Washington State University
jana@eecs.wsu.edu

Abstract

We propose a novel search-based approach for greedy coreference resolution, where the mentions are processed in order and added to previous coreference clusters. Our method is distinguished by the use of two functions to make each coreference decision: a *pruning* function that prunes bad coreference decisions from further consideration, and a *scoring* function that then selects the best among the remaining decisions. Our framework reduces learning of these functions to rank learning, which helps leverage powerful off-the-shelf rank-learners. We show that our *Prune-and-Score* approach is superior to using a single scoring function to make both decisions and outperforms several state-of-the-art approaches on multiple benchmark corpora including OntoNotes.

1 Introduction

Coreference resolution is the task of clustering a set of mentions in the text such that all mentions in the same cluster refer to the same entity. It is one of the first stages in deep language understanding and has a big potential impact on the rest of the stages. Several of the state-of-the-art approaches learn a scoring function defined over mention pair, cluster-mention or cluster-cluster pair to guide the coreference decision-making process (Daumé III, 2006; Bengtson and Roth, 2008; Rahman and Ng, 2011b; Stoyanov and Eisner, 2012; Chang et al., 2013; Durrett et al., 2013; Durrett and Klein, 2013). One common and persistent problem with these approaches is that the scoring function has to make all the coreference decisions, which leads to a highly non-realizable learning problem.

Inspired by the recent success of the $\mathcal{HC}$-Search Framework (Doppa et al., 2014a) for studying a variety of structured prediction problems (Lam et al., 2013; Doppa et al., 2014c), we study a novel approach for search-based coreference resolution called *Prune-and-Score*. $\mathcal{HC}$-Search is a divide-and-conquer solution that learns multiple components with pre-defined roles, and each of them contribute towards the overall goal by making the role of the other components easier. The $\mathcal{HC}$-Search framework operates in the space of complete outputs, and relies on the loss function which is only defined on the complete outputs to drive its learning. Unfortunately, this method does not work for incremental coreference resolution since the search space for coreference resolution consists of partial outputs, i.e., a set of mentions only some of which have been clustered so far.

We develop an alternative framework to $\mathcal{HC}$-Search that allows us to effectively learn from partial output spaces and apply it to greedy coreference resolution. The key idea of our work is to address the problem of non-realizability of the scoring function by learning two different functions: 1) a *pruning function* to prune most of the bad decisions, and 2) a *scoring function* to pick the best decision among those that are remaining. Our Prune-and-Score approach is a particular instantiation of the general idea of learning nearly-sound constraints for pruning, and leveraging the learned constraints to learn improved heuristic functions for guiding the search. The pruning constraints can take different forms (e.g., classifiers, decision-list, or ranking functions) depending on the search architecture. Therefore, other coreference resolution systems (Chang et al., 2013; Durrett and Klein, 2013; Björkelund and Kuhn, 2014) can also benefit from this idea. While our basic idea of two-level selection might appear similar to the coarse-to-fine inference architectures (Felzenszwalb and McAllester, 2007; Weiss and Taskar, 2010), the details differ significantly. Importantly, our pruning and scoring functions operate sequentially at each greedy search step, whereas in the cascades approach, the second level function makes its prediction only when the first level decision-making is done.

Summary of Contributions. The main contributions of our work are as follows. First, we motivate and introduce the *Prune-and-Score* approach to search-based coreference resolution. Second, we identify a decomposition of the overall loss of the Prune-and-Score approach into the pruning loss and the scoring loss, and reduce the problem of learning these two functions to rank learning, which allows us to leverage powerful and efficient off-the-shelf rank learners. Third, we evaluate our approach on OntoNotes, ACE, and MUC data, and show that it compares favorably to several state-of-the-art approaches as well as a greedy search-based approach that uses a single scoring function.

The remainder of the paper proceeds as follows. In Section 2, we dicuss the related work. We introduce our problem setup in Section 3 and then describe our *Prune-and-Score* approach in Section 4. We explain our approaches for learning the pruning and scoring functions in Section 5. Section 6 presents our experimental results followed by the conclusions in Section 7.

2 Related Work

The work on learning-based coreference resolution can be broadly classified into three types. First, the *pair-wise classifier* approaches learn a classifier on mention pairs (edges) (Soon et al., 2001; Ng and Cardie, 2002; Bengtson and Roth, 2008), and perform some form of approximate decoding or post-processing using the pair-wise scores to make predictions. However, the pair-wise classifier approach suffers from several drawbacks including class imbalance (fewer positive edges compared to negative edges) and not being able to leverage the global structure (instead making independent local decisions).

Second, the *global* approaches such as Structured SVMs and Conditional Random Fields (CRFs) learn a cost function to score a potential clustering output for a given input set of mentions (Mccallum and Wellner, 2003; Finley and Joachims, 2005; Culotta et al., 2007; Yu and Joachims, 2009; Haghighi and Klein, 2010; Wick et al., 2011; Wick et al., 2012; Fernandes et al., 2012). These methods address some of the problems with pair-wise classifiers, however, they suffer from the intractability of "Argmin" inference (finding the least cost clustering output among exponential possibilities) that is encountered during both training and testing. As a result, they resort to approximate inference algorithms (e.g., MCMC, loopy belief propagation), which can suffer from local optima.

Third, the *incremental* approaches construct the clustering output incrementally by processing the mentions in some order (Daumé III, 2006; Denis and Baldridge, 2008; Rahman and Ng, 2011b; Stoyanov and Eisner, 2012; Chang et al., 2013; Durrett et al., 2013; Durrett and Klein, 2013). These methods learn a scoring function to guide the decision-making process and differ in the form of the scoring function (e.g., mention pair, cluster-mention or cluster-cluster pair) and how it is being learned. They have shown great success and are very efficient. Indeed, several of the approaches that have achieved state-of-the-art results on OntoNotes fall under this category (Chang et al., 2013; Durrett et al., 2013; Durrett and Klein, 2013; Björkelund and Kuhn, 2014). However, their efficiency requirement leads to a highly non-realizable learning problem. Our Prune-and-Score approach is complementary to these methods, as we show that having a pruning function (or a set of learned pruning rules) makes the learning problem easier and can improve over the performance of scoring-only approaches. Also, the models in (Chang et al., 2013; Durrett et al., 2013) try to leverage cluster-level information implicitly (via latent antecedents) from mention-pair features, whereas our model explicitly leverages the cluster level information.

Coreference resolution systems can benefit by incorporating the world knowledge including rules, constraints, and additional information from external knowledge bases (Lee et al., 2013; Rahman and Ng, 2011a; Ratinov and Roth, 2012; Chang et al., 2013; Zheng et al., 2013; Hajishirzi et al., 2013). Our work is orthogonal to this line of work, but domain constraints and rules can be incorporated into our model as done in (Chang et al., 2013).

3 Problem Setup

Coreference resolution is a structured prediction problem where the set of mentions $m_1, m_2, \cdots, m_D$ extracted from a document correponds to a structured input x and the structured output y corresponds to a partition of the mentions into a set of clusters $C_1, C_2, \cdots, C_k$. Each mention m_i belongs to exactly one of the clusters C_j. We are provided with a training set of input-output pairs drawn from an unknown distribution $\mathcal{D}$, and the goal is to return a function/predictor from inputs to outputs. The learned predictor is evaluated against a non-negative *loss function* $L : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re^+$, $L(x, y', y)$ is the loss associated with predicting incorrect output y' for input x when the true output is y (e.g., B-Cubed Score).

In this work, we formulate the coreference resolution problem in a search-based framework. There are three key elements in this framework: 1) the *Search space* $\mathcal{S}_p$ whose states correspond to partial clustering outputs; 2) the *Action pruning function* $\mathcal{F}_{prune}$ that is used to prune irrelevant actions at each state; and 3) the *Action scoring function* $\mathcal{F}_{score}$ that is used to construct a complete clustering output by selecting actions from those that are left after pruning. $\mathcal{S}_p$ is a 3-tuple $\langle I, A, T \rangle$, where I is the initial state function, A gives the set of possible actions in a given state, and T is a predicate which is true for terminal states. In our case, $s_0 = I(x)$ corresponds to a state where every mention is unresolved, and $A(s_i)$ consists of actions to place the next mention m_{i+1} in each cluster in s_i or a NEW action which creates a new cluster for it. Terminal nodes correspond to states with all mentions resolved.

We focus on greedy search. The decision process for constructing an output corresponds to selecting a sequence of actions leading from the initial state to a terminal state using both $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$, which are parameterized functions over state-action pairs ($F_{prune}(\phi_1(s, a)) \in \Re$ and $F_{score}(\phi_2(s, a)) \in \Re$), where ϕ_1 and ϕ_2 stand for feature functions. We want to learn the parameters of both $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$ such that the predicted outputs on unseen inputs have low expected loss.

4 Greedy Prune-and-Score Approach

Our greedy *Prune-and-Score* approach for coreference resolution is parameterized by a pruning function $\mathcal{F}_{prune} : \mathcal{S} \times \mathcal{A} \mapsto \Re$, a scoring function $\mathcal{F}_{score} : \mathcal{S} \times \mathcal{A} \mapsto \Re$, and a pruning parameter $b \in [1, A_{max}]$, where A_{max} is the maximum number of actions at any state $s \in \mathcal{S}$. Given a set of input mentions $m_1, m_2, \cdots, m_D$ extracted from a document (input x), and a pruning parameter b, our Prune-and-Score approach makes predictions as follows. The search starts at the initial state $s_0 = I(x)$ (see Algorithm 1). At each non-terminal state s, the pruning function $\mathcal{F}_{prune}$ retains only the top b actions (A') from $A(s)$ (Step 3), and the scoring function $\mathcal{F}_{score}$ picks the best scoring action $a_p \in A'$ (Step 4) to reach the next state. When a terminal state is reached its contents are returned as the prediction. Figure 1 illustrates the decision-making process of our Prune-and-Score approach for an example state.

Algorithm 1 Greedy Prune-and-Score Resolver

Input: x = set of mentions $m_1, m_2, \cdots, m_D$ from a document D, $\langle I, A, T \rangle$ = Search space definition, $\mathcal{F}_{prune}$ = learned pruning function, b = pruning parameter, $\mathcal{F}_{score}$ = learned scoring function

```
1: $s \leftarrow I(x)$ // initial state
2: while not $T(s)$ do
3:     $A' \leftarrow$ Top $b$ actions from $A(s)$ according to $\mathcal{F}_{prune}$ // prune
4:     $a_p \leftarrow \arg\max_{a \in A'} \mathcal{F}_{score}(s, a)$ // score
5:     $s \leftarrow$ Apply $a_p$ on $s$
6: end while
7: return coreference output corresponding to $s$
```

We now formalize the learning objective of our Prune-and-Score approach. Let $\hat{y}$ be the predicted coreference output for a coreference input-output pair (x, y^*). The expected loss of the greedy Prune-and-Score approach $\mathcal{E}(\mathcal{F}_{prune}, \mathcal{F}_{score})$ for a given pruning function $\mathcal{F}_{prune}$ and scoring function $\mathcal{F}_{score}$ can be defined as follows.

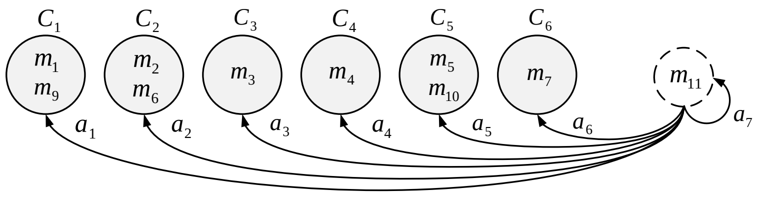
$$\mathcal{E}(\mathcal{F}_{prune}, \mathcal{F}_{score}) = \mathbb{E}_{(x,y^*) \sim \mathcal{D}} \, L(x, \hat{y}, y^*)$$

Our goal is to learn an optimal pair of pruning and scoring functions $\left(\mathcal{F}^o_{prune}, \mathcal{F}^o_{score}\right)$ that minimizes the expected loss of the Prune-and-Score approach. The behavior of our Prune-and-Score approach depends on the pruning parameter b, which dictates the workload of pruning and scoring functions. For small values of b (aggressive pruning), pruning function learning may be harder, but scoring function learning will be easier. Similarly, for large values of b (conservative pruning), scoring function learning becomes hard, but pruning function learning is easy. Therefore, we would expect beneficial behavior if pruning function can aggressively prune (small values of b) with little loss in accuracy. It is interesting to note that our Prune-and-Score approach degenerates to existing incremental approaches that use only the scoring function for search (Daumé III, 2006; Rahman and

(a) Text with input set of mentions

Ramallah (West Bank $_2$)$_1$ 10-15 (AFP$_3$) - Eyewitnesses$_4$ reported that Palestinians$_5$ demonstrated today Sunday in the West Bank$_6$ against the Sharm el-Sheikh$_7$ summit to be held in Egypt$_8$ tomorrow Monday. In Ramallah$_9$, around 500 people$_{10}$ took to **the town**$_{11}$'s streets chanting slogans denouncing the summit ...

(b) Illustration of Prune-and-Score approach



State: $s = \{C_1, C_2, C_3, C_4, C_5, C_6\}$ Actions: $A(s) = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$

Pruning step: $\underbrace{\boxed{a_2 \; a_1 \; a_7}}_{b=3}$ a_5 a_6 a_3 a_4

2.5 2.2 1.9 1.5 1.4 0.7 0.4 $\longleftarrow \mathcal{F}_{prune}$ values

$A'(s) = \{a_2, a_1, a_7\}$

Scoring step: a_1 a_2 a_7

4.5 3.1 2.6 $\longleftarrow \mathcal{F}_{score}$ values

Decision: a_1 is the best action for state s

Figure 1: Illustration of Prune-and-Score approach. (a) Text with input set of mentions. Mentions are highlighted and numbered. (b) Illustration of decision-making process for mention m_{11}. The partial clustering output corresponding to the current state s consists of six clusters denoted by $C_1, C_2, \cdots, C_6$. Highlighted circles correspond to the clusters. Edges from mention m_{11} to each of the six clusters and to itself stand for the set of possible actions $A(s)$ in state s, and are denoted by $a_1, a_2, \cdots, a_7$. The pruning function $\mathcal{F}_{prune}$ scores all the actions in $A(s)$ and only keeps the top 3 actions $A' = \{a_2, a_1, a_7\}$ as specified by the pruning parameter b. The scoring function picks the best scoring action $a_1 \in A'$ as the final decision, and mention m_{11} is merged with cluster C_1.

Ng, 2011b) when $b = \infty$. Additionally, for $b = 1$, our pruning function coincides with the scoring function.

Analysis of Representational Power. The following proposition formalizes the intuition that two functions are strictly better than one in expressive power. See Appendix for the proof.

Proposition 1. *Let $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$ be functions from the same function space. Then for all learning problems, $\min_{\mathcal{F}_{score}} \mathcal{E}(\mathcal{F}_{score}, \mathcal{F}_{score}) \geq \min_{(\mathcal{F}_{prune}, \mathcal{F}_{score})} \mathcal{E}(\mathcal{F}_{prune}, \mathcal{F}_{score})$. Moreover there exist learning problems for which $\min_{\mathcal{F}_{score}} \mathcal{E}(\mathcal{F}_{score}, \mathcal{F}_{score})$ can be arbitrarily worse than $\min_{(\mathcal{F}_{prune}, \mathcal{F}_{score})} \mathcal{E}(\mathcal{F}_{prune}, \mathcal{F}_{score})$.*

5 Learning Algorithms

In general, learning the optimal $\left(\mathcal{F}^o_{prune}, \mathcal{F}^o_{score}\right)$ pair can be intractable due to their potential interdependence. Specifically, when learning $\mathcal{F}_{prune}$ in the worst case there can be ambiguity about which of the non-optimal actions to retain, and for only some of those an effective $\mathcal{F}_{score}$ can be found. However, we observe a loss decomposition in terms of the individual losses due to $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$, and develop a stage-wise learning approach that first learns $\mathcal{F}_{prune}$ and then learns a corresponding $\mathcal{F}_{score}$.

5.1 Loss Decomposition

The overall loss of the *Prune-and-Score* approach $\mathcal{E}\left(\mathcal{F}_{prune}, \mathcal{F}_{score}\right)$ can be decomposed into *pruning loss* ϵ_{prune}, the loss due to $\mathcal{F}_{prune}$ not being able to retain the *optimal* terminal state in the search space; and *scoring loss* $\epsilon_{score|\mathcal{F}_{prune}}$, the additional loss due to $\mathcal{F}_{score}$ not guiding the greedy search to the best terminal state after pruning using $\mathcal{F}_{prune}$. Below, we will define these losses more formally.

Pruning Loss is defined as the expected loss of the *Prune-and-Score* approach when we perform greedy search with $\mathcal{F}_{prune}$ and $\mathcal{F}^*_{score}$, the optimal scoring function. A scoring function is said to be *optimal* if at every state s in the search space

$\mathcal{S}_p$, and for any set of remaining actions $A(s)$, it can score each action $a \in A(s)$ such that greedy search can reach the best terminal state (as evaluated by task loss function L) that is reachable from s through $A(s)$. Unfortunately, computing the optimal scoring function is highly intractable for the non-decomposable loss functions that are employed in coreference resolution (e.g., B-Cubed F1). The main difficulty is that the decision at any one state has interdependencies with future decisions (see Section 5.5 in (Daumé III, 2006) for more details). So we need to resort to some form of *approximate* optimal scoring function that exhibits the intended behavior. This is very similar to the dynamic oracle concept developed for dependency parsing (Goldberg and Nivre, 2013).

Let y^*_{prune} be the coreference output corresponding to the terminal state reached from input x by *Prune-and-Score* approach when performing search using $\mathcal{F}_{prune}$ and $\mathcal{F}^*_{score}$. Then the pruning loss can be expressed as follows.

$$\epsilon_{prune} = \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L\left(x, y^*_{prune}, y^*\right)$$

Scoring Loss is defined as the additional loss due to $\mathcal{F}_{score}$ not guiding the greedy search to the best terminal state reachable via the pruning function $\mathcal{F}_{score}$ (i.e., y^*_{prune}). Let $\hat{y}$ be the coreference output corresponding to the terminal state reached by *Prune-and-Score* approach by performing search with $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$ for an input x. Then the scoring loss can be expressed as follows:

$$\begin{aligned}&\epsilon_{score|\mathcal{F}_{prune}}\\&= \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L\left(x, \hat{y}, y^*\right) - L\left(x, y^*_{prune}, y^*\right)\end{aligned}$$

The overall loss decomposition of our *Prune-and-Score* approach can be expressed as follows.

$$\begin{aligned}&\mathcal{E}\left(\mathcal{F}_{prune}, \mathcal{F}_{score}\right)\\&= \underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L\left(x, y^*_{prune}, y^*\right)}_{\epsilon_{\mathbf{prune}}} +\\&\underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L\left(x, \hat{y}, y^*\right) - L\left(x, y^*_{prune}, y^*\right)}_{\epsilon_{\mathbf{score}|\mathcal{F}_{\mathbf{prune}}}}\end{aligned}$$

5.2 Stage-wise Learning

The loss decomposition motivates a learning approach that targets minimizing the errors of pruning and scoring functions independently. In particular, we optimize the overall loss of the *Prune-and-Score* approach in a stage-wise manner. We first train a pruning function $\hat{\mathcal{F}}_{prune}$ to optimize the pruning loss component ϵ_{prune} and then train a scoring function $\hat{\mathcal{F}}_{score}$ to optimize the scoring loss $\epsilon_{score|\hat{\mathcal{F}}_{prune}}$ conditioned on $\hat{\mathcal{F}}_{prune}$.

$$\begin{aligned}\hat{\mathcal{F}}_{prune} &\approx \arg\min_{\mathcal{F}_{prune}\in\mathbf{F_p}}\ \epsilon_{prune}\\ \hat{\mathcal{F}}_{score} &\approx \arg\min_{\mathcal{F}_{score}\in\mathbf{F_s}}\ \epsilon_{score|\hat{\mathcal{F}}_{prune}}\end{aligned}$$

Note that this approach is myopic in the sense that $\hat{\mathcal{F}}_{prune}$ is learned without considering the implications for learning $\hat{\mathcal{F}}_{score}$. Below, we first describe our approach for pruning function learning, and then explain our scoring function learning algorithm.

5.3 Pruning Function Learning

In our greedy *Prune-and-Score* approach, the role of the pruning function $\mathcal{F}_{prune}$ is to prune away irrelevant actions (as specified by the pruning parameter b) at each search step. More specifically, we want $\mathcal{F}_{prune}$ to score actions $A(s)$ at each state s such that the optimal action $a^* \in A(s)$ is ranked within the top b actions to minimize ϵ_{prune}. For this, we assume that for any training input-output pair (x, y^*) there exists a unique action sequence, or *solution path* (initial state to terminal state), for producing y^* from x. More formally, let $(s^*_0, a^*_0), (s^*_1, a^*_1), \cdots, (s^*_D, \varnothing)$ correspond to the sequence of state-action pairs along this solution path, where s^*_0 is the initial state and s^*_D is the terminal state. The goal is to learn the parameters of $\mathcal{F}_{prune}$ such that at each state s^*_i, $a^*_i \in A(s^*_i)$ is ranked among the top b actions.

While we can employ an *online-LaSO* style approach (III and Marcu, 2005; Xu et al., 2009) to learn the parameters of the pruning function, it is quite inefficient, as it must regenerate the same search trajectory again and again until it learns to make the right decision. Additionally, this approach limits applicability of the off-the-shelf learners to learn the parameters of $\mathcal{F}_{prune}$. To overcome these drawbacks, we apply offline training.

Reduction to Rank Learning. We reduce the pruning function learning to a rank learning problem. This allows us to leverage powerful and efficient off-the-shelf rank-learners (Liu, 2009). The reduction is as follows. At each state s^*_i on the solution path of a training example (x, y^*), we create an example by labeling optimal action $a^*_i \in A(s^*_i)$ as the only relevant action, and then try to learn a ranking function that can rank actions such that the relevant action a_i^* is in the top b actions, where b is the input pruning paramter. In other words, we have a rank learning problem, where the learner's goal is to optimize the *Precision at Top-b*. The training approach creates such an example for each state s in the solution path. The set of aggregate imitation examples collected over all the training data is then given to a rank learner (e.g., LambdaMART (Burges, 2010)) to learn the parameters of $\mathcal{F}_{prune}$ by optimizing the *Precision at Top-b* loss. See appendix for the pseudocode.

If we can learn a function $\mathcal{F}_{prune}$ that is consistent with these imitation examples, then the learned pruning function is guaranteed to keep the solution path within the pruned space for all the training examples. We can also employ more advanced imitation learning algorithms including DAgger (Ross et al., 2011) and SEARN (Hal Daumé III et al., 2009) if we are provided with an (approximate) optimal scoring function $\mathcal{F}^*_{score}$ that can pick optimal actions at states that are not in the solution path (i.e., *off-trajectory* states).

5.4 Scoring Function Learning

Given a learned pruning function $\mathcal{F}_{prune}$, we want to learn a scoring function that can pick the best action from the b actions that remain after pruning at each state. We formulate this problem in the framework of *imitation learning* (Khardon, 1999). More formally, let $(\hat{s}_0, a_0^*), (\hat{s}_1, a_1^*), \cdots, (\hat{s}_D^*, \varnothing)$ correspond to the sequence of state-action pairs along the greedy trajectory obtained by running the Prune-and-Score approach with $\mathcal{F}_{prune}$ and $\mathcal{F}^*_{score}$, the optimal scoring function, on a training example (x, y^*), where $\hat{s}_D^*$ is the best terminal state in the pruned space. The goal of our imitation training approach is to learn the parameters of $\mathcal{F}_{score}$ such that at each state $\hat{s}_i$, $a_i^* \in A'$ is ranked higher than all other actions in A', where $A' \subseteq A(\hat{s}_i)$ is the set of b actions that remain after pruning.

It is important to note that the distribution of states in the pruned space due to $\mathcal{F}_{prune}$ on the testing data may be somewhat different from those on training data. Therefore, we train our scoring function via cross-validation by training the scoring function on heldout data that was not used to train the pruning function. This methodology is commonly employed in Re-Ranking and Stacking approaches (Collins, 2000; Cohen and de Carvalho, 2005).

Our scoring function learning procedure uses cross validation and consists of the following four steps. First, we divide the training data $\mathcal{D}$ into k folds. Second, we learn k different pruners, where each pruning function $\mathcal{F}^i_{prune}$ is learned using the data from all the folds excluding the i^{th} fold. Third, we generate ranking examples for scoring function learning as described above using each pruning function $\mathcal{F}^i_{prune}$ on the data it was not trained on. Finally, we give the aggregate set of ranking examples $\mathcal{R}$ to a rank learner (e.g., SVM-Rank or LambdaMART) to learn the scoring function $\mathcal{F}_{score}$. See appendix for the pseudocode.

Approximate Optimal Scoring Function. If the learned pruning function is not consistent with the training data, we will encounter states $\hat{s}_i$ that are not on the target path, and we will need some supervision for learning in those cases. As discussed before in Section 5.1, computing an optimal scoring function $\mathcal{F}^*_{score}$ is intractable for combinatorial loss functions that are used for coreference resolution. So we employ an approximate function from existing work that is amenable to evaluate partial outputs (Daumé III, 2006). It is a variant of the ACE scoring function that removes the bipartite matching step from the ACE metric. Moreover this score is computed only on the partial coreference output corresponding to the "after state" s' resulting from taking action a in state s, i.e., $\mathcal{F}^*_{score}(s, a) = \mathcal{F}^*_{score}(s')$. To further simplify the computation, we give uniform weight to the three types of costs: 1) Credit for correct linking, 2) Penalty for incorrect linking, and 3) Penalty for missing links. Intuitively, this is similar to the correct-link count computed only on a subgraph. We direct the reader to (Daumé III, 2006) for more details (see Section 5.5).

6 Experiments and Results

In this section, we evaluate our greedy Prune-and-Score approach on three benchmark corpora – OntoNotes 5.0 (Pradhan et al., 2012), ACE 2004 (NIST, 2004), and MUC6 (MUC6, 1995) – and compare it against the state-of-the-art approaches for coreference resolution. For OntoNotes data, we report the results on both gold mentions and predicted mentions. We also report the results on gold mentions for ACE 2004 and MUC6 data.

6.1 Experimental Setup

Datasets. For OntoNotes corpus, we employ the official split for training, validation, and testing. There are 2802 documents in the training set; 343 documents in the validation set; and 345 documents in the testing set. The ACE 2004 corpus contains 443 documents. We follow the (Culotta et al., 2007; Bengtson and Roth, 2008) split in our experiments by employing 268 documents for training, 68 documents for validation, and 107 documents (ACE2004-CULOTTA-TEST) for testing. We also evaluate our system on the 128 newswire documents in ACE 2004 corpus for a fair comparison with the state-of-the-art. The MUC6 corpus containts 255 documents. We employ the official test set of 30 documents (MUC6-TEST) for testing purposes. From the remaining 225 documents, which includes 195 official training documents and 30 dry-run test documents, we randomly pick 30 documents for validation, and use the remaining ones for training.

Evaluation Metrics. We compute three most popular performance metrics for coreference resolution: MUC (Vilain et al., 1995), B-Cubed (Bagga and Baldwin, 1998), and Entity-based CEAF (CEAF$_{\phi 4}$) (Luo, 2005). As it is commonly done in CoNLL shared tasks (Pradhan et al., 2012), we employ the average F1 score (CoNLL F1) of these three metrics for comparison purposes. We evaluate all the results using the updated version[1] (7.0) of the coreference scorer.

Features. We built[2] our coreference resolver based on the Easy-first coreference system (Stoyanov and Eisner, 2012), which is derived from the Reconcile system (Stoyanov et al., 2010). We essentially employ the same features as in the Easy-first system. However, we provide some high-level details that are necessary for subsequent discussion. Recall that our features $\phi(s, a)$ for both pruning and scoring functions are defined over state-action pairs, where each state s consists of a set of clusters and an action a corresponds to merging an unprocessed mention m with a cluster C in state s or create one for itself. Therefore, $\phi(s, a)$ defines features over cluster-mention pairs (C, m). Our feature vector consists of three parts: a) mention pair features; b) entity pair features; and c) a single indicator feature to represent NEW action (i.e., mention m starts its own cluster). For mention pair features, we average the pair-wise features over all links between m and every mention m_c in cluster C (often referred to as *average-link*). Note that, we cannot employ the *best-link* feature representation because we perform offline training and do not have weights for scoring the links. For entity pair features, we treat mention m as a singleton entity and compute features by pairing it with the entity represented by cluster C (exactly as in the Easy-first system). The indicator feature will be 1 for the NEW action and 0 for all other actions.We have a total of 140 features: 90 mention pair features; 49 entity pair features; and one NEW indicator feature. We believe that our approach can benefit from employing features of the mention for the NEW action (Rahman and Ng, 2011b; Durrett and Klein, 2013). However, we were constrained by the Reconcile system and could not leverage these features for the NEW action.

Base Rank-Learner. Our pruning and scoring function learning algorithms need a base rank-learner. We employ LambdaMART (Burges, 2010), a state-of-the art rank learner from the RankLib[3] library. LambdaMART is a variant of boosted regression trees. We use a learning rate of 0.1, specify the maximum number of boosting iterations (or trees) as 1000 noting that its actual value is automatically decided based on the validation set, and tune the number of leaves per tree based on the validation data. Once we fix the hyper-parameters of LambdaMART, we train the final model on all of the training data. LambdaMART uses an internal train/validation split of the input ranking examples to decide when to stop the boosting iterations. We fixed this ratio to 0.8 noting that the performance is not sensitive to this parameter. For scoring function learning, we used 5 folds for the cross-validation training.

Pruning Parameter b. The hyper-parameter b controls the amount of pruning in our Prune-and-Score approach. We perform experiments with different values of b and pick the best value based on the performance on the validation set.

Singleton Mention Filter for OntoNotes Corpus. We employ the Illinois-Coref system (Chang et al., 2012) to extract system mentions for our OntoNotes experiments, and observe that the num-

[1]http://code.google.com/p/reference-coreference-scorers/

[2]See http://research.engr.oregonstate.edu/dral/ for our software.

[3]http://sourceforge.net/p/lemur/wiki/RankLib/

ber of predicted mentions is thrice the number of gold mentions. Since the training data provides the clustering supervision for only gold mentions, it is not clear how to train with the system mentions that are not part of gold mentions. A common way of dealing with this problem is to treat all the extra system mentions as singleton clusters (Durrett and Klein, 2013; Chang et al., 2013). However, this solution most likely will not work with our current feature representation (i.e., NEW action is represented as a single indicator feature). Recall that to predict these extra system mentions as singleton clusters with our incremental clustering approach, the learned model should first predict a NEW action while processing these mentions to form a temporary singleton cluster, and then refrain from merging any of the subsequent mentions with that cluster so that it becomes a singleton cluster in the final clustering output. However, in OntoNotes corpus, the training data does not include singleton clusters for the gold mentions. Therefore, only the large number (57%) of system mentions that are not part of gold mentions will constitute the set of singleton clusters. This leads to a highly imbalanced learning problem because our model needs to learn (the weight of the single indicator feature) to predict NEW as the best action for a large set of mentions, which will bias our model to predict large number of NEW actions during testing. As a result, we will generate many singleton clusters, which will hurt the recall of the mention detection after post-processing. Therefore, we aim to learn a singleton mention filter that will be used as a pre-processor before training and testing to overcome this problem. We would like to point out that our filter is complementary to other solutions (e.g., employing features that can discriminate a given mention to be anaphoric or not in place of our single indicator feature, or using a customized loss to weight our ranking examples for cost-sensitive training)(Durrett and Klein, 2013).

Filter Learning. The singleton mention filter is a classifier that will label a given mention as "singleton" or not. We represent each mention m in a document by averaging the mention-pair features $\phi(m, m')$ of the k-most similar mentions (obtained by ranking all other mentions m' in the document with a learned ranking function R given m) and then learn a decision-tree classifier by optimizing the F1 loss. We learn the mention-ranking function R by optimizing the recall of positive pairs for a given k, and employ LambdaMART as our base ranker. The hyper-parameters are tuned based on the performance on the validation set.

6.2 Results

We first describe the results of the learned singleton mention filter, and then the performance of our Prune-and-Score approach with and without the filter. Next, we compare the results of our approach with several state-of-the-art approaches for coreference resolution.

Singleton Mention Filter Results. Table 1 shows the performance of the learned singleton mention filter with $k = 2$ noting that the results are robust for all values of $k \geq 2$. As we can see, the learned filter improves the precision of the mention detection with only small loss in the recall of gold mentions.

	Mention Detection Accuracy		
	P	R	F1
Before-filtering	43.18% (16664/38596)	86.99% (16664/19156)	57.71%
After-filtering	79.02% (15516/19640)	80.98% (15516/19156)	79.97%

Table 1: Performance of the singleton mention filter on the OntoNotes 5.0 development set. The numerators of the fractions in the brackets show the exact numbers of mentions that are matched with the gold mentions.

Prune-and-Score Results. Table 2 shows the performance of Prune-and-Score approach with and without the singleton mention filter. We can see that the results with filter are much better than the corresponding results without the filter. These results show that our approach can benefit from having a good singleton mention filter.

Filter settings	MUC	B^3	$CEAF_{\phi 4}$	CoNLL
OntoNotes 5.0 Dev Set w. Predict Ment.				
O.S. (w.o. Filter)	66.73	53.40	44.23	54.79
P&S (w.o. Filter)	65.93	52.96	50.24	56.38
P&S (w. Filter)	**71.18**	**58.87**	**57.88**	**62.64**

Table 2: Performance of Prune-and-Score approach with and without the singleton mention filter, and Only-Score approach without the filter.

Table 3 shows the performance of different configurations of our Prune-and-Score approach. As we can see, Prune-and-Score gives better results than the configuration where we employ only the scoring function ($b = \infty$) for small values of b.

	MUC			B^3			CEAF$_{\phi4}$			CoNLL
	P	R	F1	P	R	F1	P	R	F1	Avg-F1
a. Results on OntoNotes 5.0 Test Set with Predicted Mentions										
Prune-and-Score	81.03	66.16	**72.84**	66.90	51.10	57.94	68.75	44.34	53.91	61.56
Only-Scoring	75.95	61.53	67.98	63.94	47.37	54.42	58.54	49.76	53.79	58.73
HOTCoref	67.46	74.3	70.72	54.96	62.71	**58.58**	52.27	59.4	**55.61**	**61.63**
CPL^{3}M	-	-	69.48	-	-	57.44	-	-	53.07	60.00
Berkeley	74.89	67.17	70.82	64.26	53.09	58.14	58.12	52.67	55.27	61.41
Fernandes et al., 2012	75.91	65.83	70.51	65.19	51.55	57.58	57.28	50.82	53.86	60.65
Stanford	65.31	64.11	64.71	56.54	48.58	52.26	46.67	52.29	49.32	55.43
b. Results on OntoNotes 5.0 Test Set with Gold Mentions										
Prune-and-Score	88.10	85.85	86.96	76.82	76.16	76.49	80.90	74.06	**77.33**	**80.26**
Only-Scoring	86.96	84.52	85.73	74.51	74.25	74.38	79.04	70.67	74.62	78.24
CPL^{3}M	-	-	84.80	-	-	**78.74**	-	-	68.75	77.43
Berkeley	85.73	89.26	**87.46**	78.23	75.11	76.63	82.89	70.86	76.40	80.16
Stanford	89.94	78.17	83.64	81.75	68.95	74.81	73.97	61.20	66.98	75.14
c. Results on ACE2004 Culotta Test Set with Gold Mentions										
Prune-and-Score	85.57	72.68	**78.60**	90.09	77.02	**83.04**	74.64	86.02	**79.42**	**80.35**
Only-Scoring	82.75	69.25	75.40	88.54	74.22	80.75	73.69	85.22	78.58	78.24
CPL^{3}M	-	-	78.29	-	-	82.20	-	-	79.26	79.91
Stanford	82.91	69.90	75.85	89.14	74.05	80.90	75.67	77.45	76.55	77.77
d. Results on ACE2004 Newswire with Gold Mentions										
Prune-and-Score	89.72	75.72	**82.13**	90.89	76.15	**82.87**	72.43	86.83	**78.69**	**81.23**
Only-Scoring	86.92	76.49	81.37	88.10	75.83	81.51	73.15	84.31	78.05	80.31
Easy-first	-	-	80.1	-	-	81.8	-	-	-	-
Stanford	84.75	75.34	79.77	87.50	74.59	80.53	73.32	81.49	77.19	79.16
e. Results on MUC6 Test Set with Gold Mentions										
Prune-and-Score	89.53	82.75	86.01	86.48	76.18	**81.00**	60.74	80.33	**68.68**	**78.56**
Only-Scoring	86.77	80.96	83.76	81.72	72.99	77.11	57.56	75.38	64.91	75.26
Easy-first	-	-	**88.2**	-	-	77.5	-	-	-	-
Stanford	91.19	69.54	78.91	91.07	63.39	74.75	62.43	69.62	65.83	73.16

Table 4: Comparison of Prune-and-Score with state-of-the-art approaches. Metric values reflect version 7 of CoNLL scorer.

The performance is clearly better than the degenerate case ($b = \infty$) over a wide range of b values, suggesting that it is not necessary to carefully tune the parameter b.

Pruning param. b	MUC	B^3	CEAF$_{\phi4}$	CoNLL
OntoNotes 5.0 Dev Set w. Predict Ment.				
2	69.12	56.80	56.30	60.74
3	70.50	57.89	57.24	61.88
4	71.00	58.65	57.41	62.35
5	**71.18**	**58.87**	**57.88**	**62.64**
6	70.93	58.66	57.85	62.48
8	70.12	58.13	57.37	61.87
10	70.24	58.34	56.27	61.61
20	67.97	57.73	56.63	60.78
∞	67.03	56.31	55.56	59.63

Table 3: Performance of Prune-and-Score approach with different values of the pruning parameter b. For $b = \infty$, Prune-and-Score becomes an Only-Scoring algorithm.

Comparison to State-of-the-Art. Table 4 shows the results of our **Prune-and-Score** approach compared with the following state-of-the-art coreference resolution approaches: **HOTCoref** system (Björkelund and Kuhn, 2014); **Berkeley** system with the FINAL feature set (Durrett and Klein, 2013); **CPL^{3}M** system (Chang et al., 2013); **Stanford** system (Lee et al., 2013); **Easy-first** system (Stoyanov and Eisner, 2012); and **Fernandes et al., 2012** (Fernandes et al., 2012). **Only Scoring** is the special case of our Prune-and-Score approach where we employ only the scoring function. This corresponds to existing incremental approaches (Daumé III, 2006; Rahman and Ng, 2011b). We report the best published results for CPL^{3}M system, Easy-first, and Fernandes et al., 2012. We ran the publicly available software to generate the results for Berkeley and Stanford systems with the updated CoNLL scorer. We include the results of Prune-and-Score for best b on the development set with singleton mention filter for the comparison. In Table 4, '-' indicates that we could not find published results for those cases. We see that results of the Prune-and-Score approach are comparable to or better than the state-of-the-art including Only-Scoring.

7 Conclusions and Future Work

We introduced the Prune-and-Score approach for greedy coreference resolution whose main idea is to learn a pruning function along with a scoring function to effectively guide the search. We showed that our approach improves over the methods that only learn a scoring function, and gives comparable or better results than several state-of-the-art coreference resolution systems.

Our Prune-and-Score approach is a particular instantiation of the general idea of learning nearly-sound constraints for pruning, and leveraging the learned constraints to learn improved heuristic functions for guiding the search (See (Chen et al., 2014) for another instantiation of this idea for multi-object tracking in videos). Therefore, other coreference resolution systems (Chang et al., 2013; Durrett and Klein, 2013; Björkelund and Kuhn, 2014) can also benefit from this idea. One way to further improve the peformance of our approach is to perform a search in the Limited Discrepancy Search (LDS) space (Doppa et al., 2014b) using the learned functions.

Future work should apply this general idea to other natural language processing tasks including dependency parsing (Nivre et al., 2007) and information extraction (Li et al., 2013). We would expect more beneficial behavior with the pruning constraints for problems with large action sets (e.g., labeled dependency parsing). It would be interesting and useful to generalize this approach to search spaces where there are multiple target paths from the initial state to the terminal state, e.g., as in the Easy-first framework.

Acknowledgments

Authors would like to thank Veselin Stoyanov (JHU) for answering several questions related to the Easy-first and Reconcile systems; Van Dang (UMass, Amherst) for technical discussions related to the RankLib library; Kai-Wei Chang (UIUC) for the help related to the Illinois-Coref mention extractor; and Greg Durrett (UC Berkeley) for his help with the Berkeley system. This work was supported in part by NSF grants IIS 1219258, IIS 1018490 and in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. FA8750-13-2-0033. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF, the DARPA, the Air Force Research Laboratory (AFRL), or the US government.

References

Amit Bagga and Breck Baldwin. 1998. Algorithms for scoring coreference chains. In *In The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference*, pages 563–566.

Eric Bengtson and Dan Roth. 2008. Understanding the value of features for coreference resolution. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 294–303.

Anders Björkelund and Jonas Kuhn. 2014. Learning structured perceptrons for coreference resolution with latent antecedents and non-local features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 47–57, Baltimore, Maryland, June. Association for Computational Linguistics.

Christopher Burges. 2010. From RankNet to LambdaRank to LambdaMART: An overview. *Microsoft Technical Report*, (MSR-TR-2010).

Kai-Wei Chang, Rajhans Samdani, Alla Rozovskaya, Mark Sammons, and Dan Roth. 2012. Illinois-Coref: The UI system in the CoNLL-2012 shared task. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 113–117, Jeju Island, Korea, July. Association for Computational Linguistics.

Kai-Wei Chang, Rajhans Samdani, and Dan Roth. 2013. A constrained latent variable model for coreference resolution. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 601–612.

Sheng Chen, Alan Fern, and Sinisa Todorovic. 2014. Multi-object tracking via constrained sequential labeling. In *To appear in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

William W. Cohen and Vitor Rocha de Carvalho. 2005. Stacked sequential learning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 671–676.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 175–182.

Aron Culotta, Michael L. Wick, and Andrew McCallum. 2007. First-order probabilistic models for coreference resolution. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, pages 81–88.

Hal Daumé III. 2006. *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, University of Southern California, Los Angeles, CA.

Pascal Denis and Jason Baldridge. 2008. Specialized models and ranking for coreference resolution. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 660–669.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. 2014a. HC-Search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research (JAIR)*, 50:369–407.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. 2014b. Structured prediction via output space search. *Journal of Machine Learning Research (JMLR)*, 15:1317–1350.

Janardhan Rao Doppa, Jun Yu, Chao Ma, Alan Fern, and Prasad Tadepalli. 2014c. HC-Search for multi-label prediction: An empirical study. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Greg Durrett and Dan Klein. 2013. Easy victories and uphill battles in coreference resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1971–1982.

Greg Durrett, David Leo Wright Hall, and Dan Klein. 2013. Decentralized entity-level modeling for coreference resolution. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, pages 114–124.

Pedro F. Felzenszwalb and David A. McAllester. 2007. The generalized A* architecture. *Journal of Artificial Intelligence Research (JAIR)*, 29:153–190.

Eraldo Rezende Fernandes, Cícero Nogueira dos Santos, and Ruy Luiz Milidiú. 2012. Latent structure perceptron with feature induction for unrestricted coreference resolution. International Conference on Computational Natural Language Learning (CoNLL), pages 41–48.

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

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

Aria Haghighi and Dan Klein. 2010. Coreference resolution in a modular, entity-centered model. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*.

Hannaneh Hajishirzi, Leila Zilles, Daniel S. Weld, and Luke S. Zettlemoyer. 2013. Joint coreference resolution and named-entity linking with multi-pass sieves. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 289–299.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning Journal (MLJ)*, 75(3):297–325.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*.

Roni Khardon. 1999. Learning to take actions. *Machine Learning Journal (MLJ)*, 35(1):57–90.

Michael Lam, Janardhan Rao Doppa, Xu Hu, Sinisa Todorovic, Thomas Dietterich, Abigail Reft, and Marymegan Daly. 2013. Learning to detect basal tubules of nematocysts in sem images. In *ICCV Workshop on Computer Vision for Accelerated Biosciences (CVAB)*. IEEE.

Heeyoung Lee, Angel X. Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2013. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4):885–916.

Qi Li, Heng Ji, and Liang Huang. 2013. Joint event extraction via structured prediction with global features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 73–82.

Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331.

Xiaoqiang Luo. 2005. On coreference resolution performance metrics. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 25–32, Stroudsburg, PA, USA. Association for Computational Linguistics.

Andrew Mccallum and Ben Wellner. 2003. Toward conditional models of identity uncertainty with application to proper noun coreference. In *Proceedings of Neural Information Processing Systems (NIPS)*, pages 905–912. MIT Press.

MUC6. 1995. Coreference task definition. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, pages 335–344.

Vincent Ng and Claire Cardie. 2002. Improving machine learning approaches to coreference resolution. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, pages 104–111.

NIST. 2004. The ACE evaluation plan.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Proceedings of the Joint Conference on EMNLP and CoNLL: Shared Task*, pages 1–40.

Altaf Rahman and Vincent Ng. 2011a. Coreference resolution with world knowledge. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, pages 814–824.

Altaf Rahman and Vincent Ng. 2011b. Narrowing the modeling gap: A cluster-ranking approach to coreference resolution. *Journal of Artificial Intelligence Research (JAIR)*, 40:469–521.

Lev-Arie Ratinov and Dan Roth. 2012. Learning-based multi-sieve co-reference resolution with knowledge. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP) Conference*, pages 1234–1244.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research - Proceedings Track*, 15:627–635.

Wee Meng Soon, Daniel Chung, Daniel Chung Yong Lim, Yong Lim, and Hwee Tou Ng. 2001. A machine learning approach to coreference resolution of noun phrases.

Veselin Stoyanov and Jason Eisner. 2012. Easy-first coreference resolution. In *Proceedings of International Conference on Computational Linguistics (COLING)*, pages 2519–2534.

Veselin Stoyanov, Claire Cardie, Nathan Gilbert, Ellen Riloff, David Buttler, and David Hysom. 2010. Coreference resolution with reconcile. In *Proceedings of Association of Computational Linguistics (A-CL) Conference*, pages 156–161.

Marc B. Vilain, John D. Burger, John S. Aberdeen, Dennis Connolly, and Lynette Hirschman. 1995. A model-theoretic coreference scoring scheme. In *MUC*, pages 45–52.

David Weiss and Benjamin Taskar. 2010. Structured prediction cascades. *Journal of Machine Learning Research - Proceedings Track*, 9:916–923.

Michael L. Wick, Khashayar Rohanimanesh, Kedar Bellare, Aron Culotta, and Andrew McCallum. 2011. SampleRank: Training factor graphs with atomic gradients. In *Proceedings of International Conference on Machine Learning (ICML)*.

Michael L. Wick, Sameer Singh, and Andrew McCallum. 2012. A discriminative hierarchical model for fast coreference at large scale. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, pages 379–388.

Yuehua Xu, Alan Fern, and Sung Wook Yoon. 2009. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research (JMLR)*, 10:1571–1610.

Chun-Nam John Yu and Thorsten Joachims. 2009. Learning structural SVMs with latent variables. In *Proceedings of International Conference on Machine Learning (ICML)*.

Jiaping Zheng, Luke Vilnis, Sameer Singh, Jinho D. Choi, and Andrew McCallum. 2013. Dynamic knowledge-base alignment for coreference resolution. In *Conference on Computational Natural Language Learning (CoNLL)*.