Latent Tree Structure Learning for Cross-Document Coreference Resolution

by

Eric Shyu

B.S., Massachusetts Institute of Technology (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Signature redacted

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2014

Signature redacted

Certified by .....
Leslie P. Kaelbling
Panasonic Professor of Computer Science and Engineering
Thesis Supervisor

Signature redacted

Accepted by .....
Albert J. Meyer
Chairman, Masters of Engineering Thesis Committee

Latent Tree Structure Learning for Cross-Document Coreference Resolution

by

Eric Shyu

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2014, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

Cross Document Coreference Resolution (CDCR) is the problem of learning which mentions, coming from several different documents, correspond to the same entity. This thesis approaches the CDCR problem by first turning it into a structure learning problem. A latent tree structure, in which leaves correspond to observed mentions and internal nodes correspond to latent sub-entities, is learned. A greedy clustering heuristic can then be used to select subtrees from the learned tree structure as entities.

As with other structure learning problems, it is prudent to envoke Occam's razor and perform regularization to obtain the simplest hypothesis. When the state space consists of tree structures, we can impose a bias on the possible structure. Different aspects of tree structure (i.e. number of edges, depth of the leaves, etc.) can be penalized in these models to improve the generalization of thes models.

This thesis draws upon these ideas to provide a new model for CDCR. To learn parameters, we implement a parameter estimation algorithm based on existing stochastic gradient-descent based algorithms and show how to further tune regularization parameters. The latent tree structure is then learned using MCMC inference. We show how structural regularization plays a critical role in the inference procedure. Finally, we empirically show that our model out-performs previous work, without using a sophisticated set of features.

Thesis Supervisor: Leslie P. Kaelbling
Title: Panasonic Professor of Computer Science and Engineering

Acknowledgments

First and foremost I want to thank Professor Leslie Kaelbling. Before Leslie took me as her student, she was an extremely influential figure in helping me build my current interests, from teaching my first ever class in computer science (6.01) and my very first exposure to machine learning (6.867). Whenever I had any questions about anything, Leslie has always been prompt to answer. Under Leslie's guidance I have learned how to dig really deep into a problem and not allow any questions go unanswered. I am truly grateful to have had Leslie as my advisor.

Next I must thank the people at Diffeo. Through their guidance and support, my work there as a summer intern was able to turn into a full thesis. Most of the code I developed for this thesis was built off of the Diffeo code base. Everything about this thesis would not be possible without them. I want to thank Jay Baxter and Andrew Smith for being amazingly fun interns to code with and always being there to bounce off ideas with me. Thanks go to Nilesh Tripuranei for the amazingly insightful and fun lunch conversations and all the things he has shown me over the past year. Many of the initial ideas behind this thesis (such as the Metropolis Hastings proposal distribution and the greedy clustering heuristic used to select entities) were first developed by Dan Roberts and Max Kleiman-Weiner. I want to thank Steve Bauer for his always-helpful guidance around Diffeo's code base and advice about the future. I want to thank John Frank for his support, having faith in me, and giving me the opportunity to work at Diffeo and this thesis. Watching him lead and build Diffeo team over the past year really has been inspirational.

Finally, I want to thank my mother and father. Thank you for your constant support and love. I could not have made it this far without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Much of today's digital information exists in the form of unstructured textual web pages, news articles, reports, and documents. From such unstructured text, a common task is to extract the mentioned entities (people, places, organizations, etc.) and subsequently identify the relevant contextual mentions about them. This known as the *coreference resolution* task, and it has many applications in question answering, machine-translation [11], text summarization [26], and author disambiguation [18].

An *entity* can be any object, place, organization, or person, such as the basketball player Michael Jordan. An entity *mention* is a textual string that refers to the entity. The coreference task is to identify entities and mentions, and then group all mentions of the same entity together. Figure 1-1 gives a simple example of the coreference resolution problem.

Looking at Figure 1-1, a human can readily identify that: mentions 1 and 3 are the same entity (Michael Jordan the basketball legend), mention 2 is a separate entitity (Michael Jordan, the statistician and professor at Berkley), mention 4 is a spearate entity (Michael Jackson, the late pop singer), and mention 5 is its own entity (the president Barack Obama).

One strategy might be to first extract the entity names from each mention. It seems pretty clear that "Barack Obama" is probably totally different than "Michael

1. " **Michael Jordan** played on two Olympic gold medal-winning American basketball teams"
2. " **Michael Jordan** published a paper about scalable statistical inference algorithms."
3. " Legendary **MJ** appeared in the 2001 basketball movie Space Jam."
4. " **MJ** was the legendary king of pop."
5. " **Barack Obama** addressed the issue of Syria today"

Figure 1-1: Example mentions for a coreference problem. Mentions of the entities are shown in bold. In this example, the correct coreferent partitioning is $\{1,3\},\{2\},\{4\},\{5\}$.

Jordan" or "MJ". Then, one might use contextual words in the mention to disambugate between similarly-named entities.

Our goal is to translate this human intuition into a systematic algorithm.

1.2 Cross-Document Coreference Resolution

Many successful approaches for coreferencing mentions obtained from a single document (*within-document coreference resolution*) have been previously developed [11, 22]. Complications arise when allowing mentions to be drawn from several different documents. For example, while it is reasonable to assume that each mention of "Michael Jordan" in the same news article is likely to be about the same person, as Figure 1-1 shows, across multiple documents this is no longer the case. This is known as the *cross-document* coreference resolution (CDCR) problem. Much of the recent work on coreference systems has been dedicated to creating scalable solutions for the CDCR task [20, 22, 25, 3].

The first step is generally to start with *within-document coreference resolution*, where all all the mentions are extracted from the same document. Named-entity Recognition (NER) algorithms can easily identify the entity mentions within a document, decide if these mentions are referring to the same individual, and create

coreference chains for each unique entity. Several publicly available systems exist for the NER task: OpenNLP, Stanford NER, Lingpipe NER, and BBN SERIF [2].

After identifying the entities and linking mentions into coreference chains, the *cross-document coreference resolution* (CDCR) problem is to determine which of these chains are in fact referencing same entity. To accomplish this task, several features such as the bag of contextual words and topics found using Latent Dirichlet Allocation [4], names, and biographical data can be used. More specialized features can be introduced for more specific domains. For example, in the author coreference problem [18], possible features include headers, paper titles, topic keywords of the papers, etc. Several different algorithms have been proposed to leverage these features in performing CDCR.

In this thesis, we take inspiration from many of these existing approaches, to create a new approach based on learning a single latent tree structure over mentions. Entities can then be extracted by partitioning the tree into subtrees, each corresponding to an entity. This approach allows us to view the coreference problem as a form of structure learning, from which we borrow the idea of invoking Occam's razor to penalize more complex structures. Although we will not focus on engineering better features for CDCR, we develop a system for learning relative weights between features from the data.

1.3 Outline of Thesis

The remainder of this thesis is structured as follows.

Chapter 2 discusses some of the previous work related to performing cross-document coreference resolution. This chapter summaries some of the current start-of-the-art and our evaluation metrics.

Chapter 3 is background material, not directly related to coreference, that is necessary to motivate and understand our method. It will summarize the the ideas we are using from structure learning, the Markov Chain Monte Carlo inference, and the algorithms that we use for parameter estimation.

Chapter 4 introduces our new approach for expressing coreference as a tree structure learning problem. We describe the motivation, our factor graph formulation, and how we invoke Occam's razor to penalize more complex structures.

Chapter 5 describes the specific implementations and methods used to train parameters and perform inference with our model.

Chapter 6 evalutes our method on real corpora. We provide some new analysis techniques to study the role structural regularization plays to make our model generalize well.

Chapter 7 wraps up the thesis, concluding with ideas for future work. In particular it describes how our approach can be made massively parallel as well as adapted into an online setting.

Chapter 2

Related Work

In this chapter we describe the common and previously most successful approaches for tackling the CDCR task.

2.1 Pairwise Coreference Models

Coreference resolution is often framed as a pairwise classification task. Namely, given all pairs of mentions, the question is to determine whether or not they belong to the same entity. These methods often apply some sort of greedy clustering using some pairwise simliarity function based on the mentions' contextual information in order to make these pairwise decisions.

For example, Bagga and Baldwin [1] exprese mentions as vectors of contextual features (such as TF/IDF-weighted bag-of-words) and applies greedy agglomerative clustering with single linkage. Mayfield et. al [15] describe the pairwise coreference decisions as edges in a graph. The authors start with a complete entity graph and elimininate entity pair edges that have an SVM output weight less than 0.95; the connected components are then taken as entities.

Because these algorithms require deciding if each mention-pair is about the same entity, such approaches have complexity $\Omega(n^2)$ in the number of mentions. This is too slow for a CDCR system running on thousands or millions of documents, which can have orders of magnitude more mentions. Furthermore, because the pairwise

framework inherently only considers relationship between different mention pairs, it does not consider the aggregate properties of each entity.

2.2 Graphical Models Approaches

Markov Random Fields and Conditional Random Fields are able to address some of the problems with pairwise coreference models, because it is possible to encode transitivity constraints (e.g. if mention m_1 is coreferent with m_2 and m_2 is coreferent with m_3, then m_1 and m_3 are coreferent) and functions over sets of mentions as graph potentials. Recent work [23, 19, 25, 8] have shown that models that are able to express these aggregrate properties can potentially perform better coreference.

These approaches thus embed the coreference decisions in a graphical model and turn the task into performing inference on the graphical model. Because these graphical models are large, exact inference is not possible but Markov Chain Monte Carlo inference has been shown to often work well in practice [22, 25].

2.3 Online Algorithms

Most of the previously mentioned approaches are restricted to offline scenarios where documents are provided in advance. However for a practical modern CDCR system, it is reasonable to expect a high volume of streaming text, from which we wish to extract the mentions and coreference them.

[20] develops a streaming algorithm in which each document is processed exactly one time. This algorithm either adds points to existing clusters or creates new clusters depending on a mention's similarity with the clusters. In order to avoid comparing a mention with all the existing clusters, the algorithm implements a high-recall filter to compare each mention with only k potentially similar clusters.

2.4 Hierarchical Models

Sameer et al. [22] first developed the hierarchical coreference framework most similar to our approach. In their model, a two-tiered hierarchy is used in order in order to organize proposals for Metropolis Hastings sampling. This hierarchy is fixed and shallow.

Wick et al. [25] recursively decompose each entity is into an arbitrarily deep tree of sub-entities. The leaves of each entity tree correspond to the observed mentions, while the internal nodes correspond to latent sub-entities and the tree root corresponds to the entity. Each sub-entity serves as a summary of all its children. While in principle, the summary attributes of a node can be inferred from the summary attributes of its children, the authors accomplish this by just aggregating vector (bag) features of the children. Finally, the likelihood of an entity tree configuration is a function pairwise compatability functions between each node and its parent, as well as unitary functions measuring the diversity of a single node.

Wick et al. [25] then uses a Metropolis Hastings sampler to propose moving sub-trees between entities, creating new entities, deleting sub-entities, etc. In order to further encourage better proposals, the authors use "canopy" functions (which serve as high-recall filters) so that two subtrees are sampled only if they contain mentions of entities with similar first initial and last name. While the state space for inference over the tree structure of an entity is much larger than the state space of mention-pair decisions, the authors show that this model is often able to achieve similar F_1 scores as pairwise coreference several orders of magnitude faster, in number of samples and computation time per sample.

In section 4.3, we will compare these works with our approach.

2.5 Evaluation Metrics

As with most information retrieval tasks, the primary concerns in the CDCR problem are precision (the fraction of retrieved instances that are relevant) and recall (the

fraction of relevant instances that are retrieved). Given precision and recall, the harmonic mean

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

is generally used as the final metric.

Several different ways of computing these quantities have been proposed, such as the ACE-values, CEAF, and MUC6 measures [2]. For this thesis we will be concerned with the Pairwise and B-CUBED methods for computing F_1 score.

2.5.1 Pairwise F_1:

Related work [25, 24] has often used the "pairwise F_1" score, which given a partitioning of the mentions, looks at the set of all mention pairs. A true-positive pair is such that two mentions are in the same partition and of the same entity. A false-positive pair is such that two mentions are in the same partition and of different entities. Similarly, we can define true-negative and false-negative pairs. Using these statistics, precision and recall statistics can be computed as usual.

Despite being a fairly widely-cited metric [24, 16, 14], we could not find the algorithm for computing it in the literature. Because the pairwise F_1 score is a little subtle to compute, our implementation of the pairwise-F_1 algorithm is shown in Figure 2-1. Despite appearing fairly complicated, the pairwise F_1 score can be computed quite efficiently if each hypothesized entity $\hat{E}_m$ is represented as a dictionary of counts for each true entity it contains.

2.5.2 B-CUBED F_1:

In present literature, the most common evaluation metric in published research is the B-CUBED scoring algorithm [1]. This works by computing a precision and recall for each individual mention and taking the total precision and recall to be weighted averages of these.

The B-CUBED algorithm is defined in Figure 2-2. For each mention B-CUBED measures the presence or absence of other (falsely) coreferent mentions in the hypoth-

1. Let $E(\hat{E}_m)$ be the set of truly different entities in $\hat{E}_m$. Define $count_m(e_i)$ to be the number of mentions of entity e_i in $\hat{E}_m$

2. For each hypothesized entity $\hat{E}_m$:

   (a) Compute true positives: the number of pairs of mentions in $\hat{E}_m$ that belong to the same entity

   $$TP(\hat{E}_m) = \sum_{e_i \in E(\hat{E}_m)} \binom{count_m(e_i)}{2}$$

   (b) Compute false positives: the number of pairs mentions in $\hat{E}_m$ that belong to the different entities

   $$FP(\hat{E}_m) = \frac{1}{2} \sum_{e_i, e_j \in E(\hat{E}_m)} count_m(e_i) count_m(e_j)$$

3. For each pair of hypothesized entities $\hat{E}_m, \hat{E}_n$:

   (a) Compute the number of false negatives: the number of pairs of mentions $m_i \in \hat{E}_m$ and $m_j \in \hat{E}_n$ that are hypothesized to be in different entities but, belong to the same entity

   $$FN(\hat{E}_m, \hat{E}_n) = \sum_{e_i \in \hat{E}_m \cup \hat{E}_n} count_m(e_i) count_n(e_i)$$

4. Aggregate the number of pairs of true positives, false positives, and false negatives:

   $$TP = \sum_m TP(\hat{E}_m)$$
   $$FP = \sum_m FP(\hat{E}_m)$$
   $$FN = \sum_{m,n} FN(\hat{E}_m, \hat{E}_n)$$

5. Compute F_1 score as

   $$\frac{TP}{2TP + FN + FP}$$

Figure 2-1: Our implementation of the Pairwise F_1 Score Algorithm

1. For each entity mention m:

   (a) Let E_m be its true entity cluster and $\hat{E}_m$ be its hypothesized entity cluster. Note that each m belongs to only one E_m and only one $\hat{E}_m$.

   (b) Compute the precision and recall score for E_m and $\hat{E}_m$ as:

$$P_m = \frac{|E_m \cup \hat{E}_m|}{|\hat{E}_m|}$$

$$R_m = \frac{|E_m \cup \hat{E}_m|}{|E_m|}$$

2. Compute overall precision and recall:

$$P = \frac{1}{M} \sum_{m=1}^{M} P_m$$

$$R = \frac{1}{M} \sum_{m=1}^{M} R_m$$

Taking the harmonic mean yields statistics such as B-CUBED F_1 score.

Figure 2-2: B-CUBED F_1 Score Algorithm

esized clustering. Intuitively, it is treating precision and recall as mention-weighted averages of how well the hypothesized clusters capture other mentions of the same entity.

Chapter 3

Background Material

In this chapter we review the background behind the paradigms and algorithms that drive our model. The following are general methods that are not specific to the CDCR task.

3.1 Structure Learning

A typical problem in bio-informatics and computer vision is to learn latent tree graphical models over random variables. The three challenging goals in this domain are determining:

1. The number of latent variables
2. The conditional dependencies between the variables
3. The parameters characterizing the relationships between the variables

The first two goals are often described in terms of graphical models, such as Markov Random Fields or factor graphs. Using a graphical model representation, the second goal is the same as determining the structure of the graphical model. If we have determined that there are N latent variables, then we have $2^{N(N-1)/2}$ different models (corresponding to choosing to include each of the $\binom{N}{2}$ edges in the graph) to choose.

One obvious way to select the best model is to pick the one that has the highest likelihood corresponding to the data. However, this approach will almost always

choose more complicated models with more edges and more latent variables. In all cases, structural regularization is important to prevent over-complexity and improve generalization. To this end, one might use penalizations such as Bayes Information Criterion or Akaike information criterion to select the simplest graphical models that represent the data well.

There are methods that explicitly try to learn latent tree structures. The Chow-Liu algorithm [7] is one such well-known method. To add some more structure to the problem, hierarchical latent class (HLC) models require observed nodes to be leaves and have learning algorithms based on introducing new hidden nodes or replacing edges [6].

Phylogenetics researchers have also extensively studied different ways of learning latent tree structures to model evolutionary history of DNA sequences. Several model have defined probability distributions over trees and used MCMC to perform inference [13]. However, these models use very different data than the coreference problem, hence the form of the models tend to be very different.

3.2 Markov Chain Monte Carlo Inference

Markov Chain Monte Carlo (MCMC) methods are commonly used to draw samples $x \sim p(\cdot;\theta)$, where the distribution

$$p(\cdot;\theta) = \frac{f(\cdot;\theta)}{Z(\theta)}$$

is only known up to $f(\cdot;\theta)$ and the partition function $Z(\theta) = \int f(x;\theta)\,dx$ is intractable to compute.

In particular, the Metropolis Hastings algorithm first draws samples from an ergodic Markov chain $q(\cdot|\cdot)$, which serves as a *proposal distribution*, and accept samples in such a manner that results in the accepted samples following a new ergodic Markov chain that has $p(\cdot;\theta)$ as its stationary distribution. Specifically if the current state is

x and the proposed next state is x', we compute an acceptance ratio

$$\alpha_T(x, x') = \min\left(1, \frac{f(x';\theta)}{f(x;\theta)}\left(\frac{q(x|x')}{q(x'|x)}\right)^{1/T}\right)$$

As the parameter $T \to 0$, Metropolis Hastings accepts new proposals x' that increase the likelihood. Thus, lowering the parameter T to zero allows us to use Metropolis Hastings for performing MAP inference. The specific way that T is lowered to zero is often called the *cooling schedule*. The Metropolis Hastings algorithm for performing MAP inference is shown in Figure 3-1.

1. Initialize x
2. For $i = 1, 2, \ldots$
   (a) Draw $x' \sim q(\cdot|x)$.
   (b) Decrease T using cooling schedule
   (c) Flip a coin with bias $\alpha_T(x, x')$
      i. If heads, let the new state $x = x'$
      ii. If tails, let new state $x = x$, the old state

Figure 3-1: Metropolis Hastings Algorithm for Inference

MCMC-based methods are able to quickly explore high-dimensional state spaces, which make them very attractive for many modern problems.

3.3 Stochastic Gradient Descent Estimation

We now turn to the problem of estimating parameters for our models. In practice, some of our mentions may be labeled (usually by hand annotation) with the identity of its correct entity. Thus, for a particular corpus we can try to learn parameters that, on average, produce the best coference decisions.

In this section we outline two different methods based on stochastic gradient descent to learn parameters from data. Recall that basic stochastic gradient descent

update is

$$\theta_t = \theta_{t-1} + \eta \hat{\nabla} f(\theta_{t-1})$$

where f is our objective function, which depends on all of the data, and $\hat{\nabla} f$ is an approximation of the gradient of f using a single random data point. In order for our inference to be efficient we will turn to MCMC. Thus, we seek a parameter estimation method that is compatible with such approximate inference. Constrastive Divergence [9] and SamplerRank [24] both are designed to work well with MCMC inference.

Both approaches are very similar, but optimize different objective functions. As we shall see, Constrastive Divergence computes biased gradients using the ground truth configuration, while SampleRank uses an objective function rankings among neighboring states.

3.3.1 Contrastive Divergence

The idea of Contrastive Divergence (CD) is to approximate the gradient of the likelihood function with respect to the model parameters, by sampling in the neighborhood of the ground-truth state.

Suppose our model is

$$p(y;\theta) \propto \exp\{-E(y,\theta)\}$$

where y is our current state and θ is the parameter we are trying to optimize. CD approximates the gradient of the log-likelihood as

$$\frac{\delta \log p(y;\theta)}{d\theta} \approx \left[\frac{\delta E(y;\theta)}{\delta\theta}\right]_1 - \left[\frac{\delta E(y;\theta)}{\delta\theta}\right]_0$$

where the 1 subscript denotes a state y' that is one MCMC walk-step away from y, and the 0 subscript denotes te state y. Because MCMC can approximate the expectation of a function, this yields an algorithm for approximating the gradient. With the gradient approximiation in hand, we can use basic gradient descent to find the maximum likelihood parameters. If state y^* is the ground truth state, then contrastive divergence learns θ^* so that the $y^* = \arg\max_y p(y;\theta^*)$ (hence that y^* is

the maximum likelihood state of $p(\cdot;\theta^*)$.

Typically only one walk-step from the ground truth is needed to reach convergence. However, since only one step from the optimum configuration, the gradient approximation is biased. Despite using such a biased estimate of the gradient, contrastive divergence using only one step emprically performs well [5].

3.3.2 SampleRank

SampleRank is designed for learning parameters of log-linear models

$$p(y|x) \propto \exp\{\theta \cdot \phi(x,y)\} = \exp\left\{\sum_i \theta_i \phi_i(x,y)\right\}$$

where y is the latent state, x is the observed inputs, ϕ_i are sufficient statistics computed over different features from observed data x, and w_i are the corresponding log-linear weights.

Rather than approximating maximum likelihood, SampleRank [24] uses a user-provided objective function. At a high level, SampleRank works by generating pairs of neighboring samples using an MCMC chain and updates model parameters if their model rankings disagrees with the objective function ranking (hence the name of the algorithm). Such a disagreement between the model and the objective is called a *constraint violation.* The implementation of SampleRank can be embedded within the walk-step of an MCMC inference algorithm. The SampleRank algorithm is shown in Algorithm 3-2.

The objective function does not need any special properties, such as convexity. First, we require it to output a deterministic ranking between pairs of neighboring states. Second, it should be possible to make a sequence of proposals towards the ground truth state without decreasing the objective [24]. A smart objective function can take advantage of domain-specific signals. For example, a coreference task could use F_1 score as the objective.

Using pairs of neighboring states (meaning that they are within one-step of each other in the proposal distribution) can allow changes in the objective function and

1. Let:
   (a) $F(\cdot)$ be an objective function
   (b) $q(\cdot|\cdot)$ be proposal distribution from Metropolis Hastings
   (c) η be learning rate
2. Initialize state x, $\theta^{(0)} = 0$
3. For $i = 1, 2, \ldots$
   (a) Attempt an Metropolis Hastings proposal by drawing $x' \sim q(\cdot|x)$
   (b) Define

$$x^+ = \arg\max_{z \in \{x, x'\}} F(z)$$
$$x^- = \arg\min_{z \in \{x, x'\}} F(z)$$
$$\nabla = \phi(x^+) - \phi(x^-)$$

   (c) If $\theta^{(i)} \cdot \nabla < F(x^+) - F(x^-)$:
      i. Update $\theta^{(i+1)} = \theta^{(i)} + \eta\nabla$
   (d) If Metropolis Hastings accepts x'
      i. Update $x = x'$

Figure 3-2: SampleRank Algorithm

likelihood function to be computed efficiently. Neighboring states will differ only by small local changes, so it is possible to update the objective and likelihood efficiently, without recomputing them from scratch.

Empirically it has been shown that SampleRank can be much faster than CD [24]. Additionally, SampleRank seems to have little trouble scaling to hundreds or even thousands of features. However we use both Samplerank and CD in our tests for benchmarking purposes.

Chapter 4

Latent Tree Structure Learning for Coreference

4.1 Motivation

4.1.1 Hierarchical Entity Trees

An important observation made by Sameer, Wick, et al. [22, 25] was that it is possible to reduce the time complexity inherent in the pairwise approach by changing the underlying model. In their hierarchical coreference model, entities are recursively structured into sub-entities. Each entity is realized as a hierarchical tree, where leaves correspond to mentions and internal nodes correspond to latent sub-entitites. These latent sub-entities summarize the the attributes of their children. Thus, conditioned on observing a sub-entity ought to be sufficient to make statements about its contents.

For Sameer, Wick, et al. [22, 25], the sub-entity structures serve the purpose of allowing block moves during MCMC inference. In our approach, the hierarchical sub-entity structure is a crucial thing that we are trying to learn. So unlike these previous works, the focus of our approach is to learn a tree structure that captures the hierarchy in the mentions; obtaining entity clusters for coreference is then accomplished using the tree structure.

Returning to the example of Figure 1-1, we might build the tree shown in Figure 4-

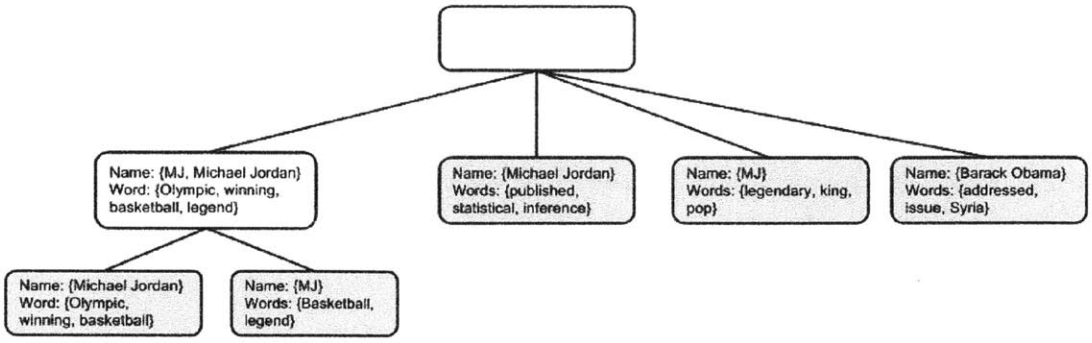


Figure 4-1: A hierarchical tree to solve the corefrence problem posed by Figure 1-1. Filled vertices indicate mentions, while unfilled vertices summarize their descendants.

1. Note we have combined the two Michael-Jordan-basketball-player mentions into one subtree and summarized them with their common parent. Thus, instead of comparing the Barack Obama to both of the Michael-Jordan-basketball-player mentions, we can compare it to only its summarizing internal vertex. Thus, in general, time complexity of comparing the contents of two entities is no longer necessarily dependent on the number of its mentions. This allows the coreference problem to become computationally tractable and allows block moves, which is the goal of [22, 25].

However, a hierarchical tree potentially has other useful properties. Consider if we instead made the tree in Figure 4-2. This tree has a subtree which contains both the Michael-Jordan-basketball-player and Michael Jackson entities. The additional hierchical structure over entities of this tree allows us to see that these two entities might be related (the common use of "legend" indicates that both are celebrities).

In our model, we allow an unconstrained hierarchy over entities as well as sub-entities, so the goal of the inference task is not to learn what the best entity partitions are, but rather to learn what the best latent tree structure over observed mention leaves is. While this seems like adding an unnecessary level of complication for the CDCR task, we shall see that this approach performs comparably with previous work.

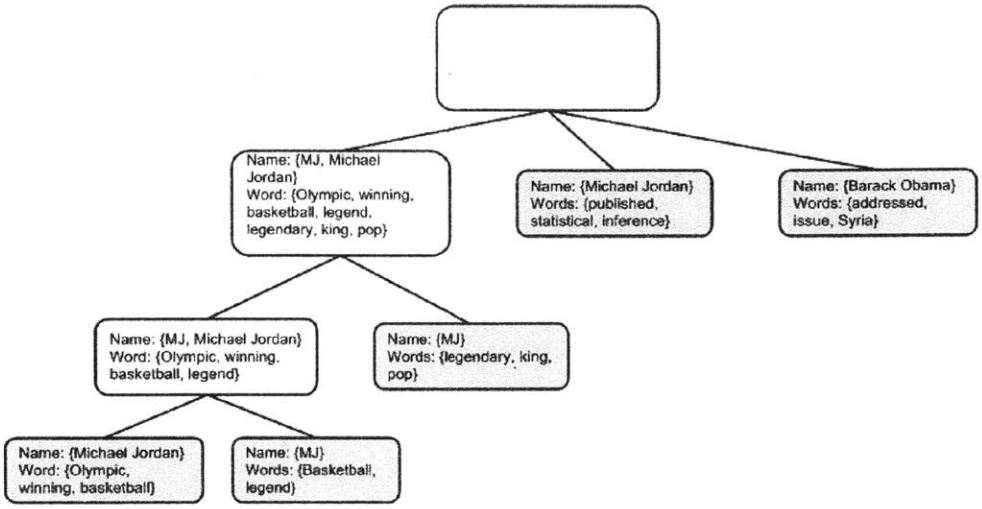


Figure 4-2: A second hierarchical tree to solve the coreference problem posed by Figure 1-1. Now there exists a subtree containing both MJ the legendary basketball player and MJ the legendary pop star. Both are related by words indicating they are popular celebrities, which is relevealed by the tree structure.

4.2 The Model

Our goal is to construct a hierarchical tree over the mentions from which the true entities can be easily recovered. Notably different from [25], we explicitly are modeling a potential hierarchy over entities as well as sub-entities. Thus with this approach, it is not completely clear which internal nodes correspond to entities or sub-entities during inference.

In our model, the state of each tree node consists of a set of the feature vectors, such as bag-of-words or bag-of-names. Thus our model uses the same vector space model used by most previous work [1, 20, 3, 22, 25]. The leaves of the tree are treated as being observed, while the sub-entities nodes are treated as latent.

4.2.1 Constraints

While our approach formulates CDCR as a structure learning problem, we add two constraints to the structure and states of our graphical model in order to make inference more tractable.

1. (Structural Constraint) **Each internal node must have at least two children.** This prevents arbitrarily long linear chains of latent sub-entities that correspond to the exact same entity.

2. (State Constraint) **The state of each latent node is the sum of the states of its children.** This is a simple way of summarizing the states of all the terminal mentions that are descended from a latent node. Previous work [22, 25] have also used this constraint with success.

The second constraint relieves us from having to estimate the states of the latent nodes, which would be necessary in normal structure learning problems. Thus our model is purely concerned with learning a tree structure that fits the corpus well.

4.2.2 Factor Graph Formulation

Let $\mathcal{M}$ be the set of observed mentions. We use $\mathcal{T}_{\mathcal{M}}$ to denote the set of all trees with $\mathcal{M}$ as mentions. We would like to find a tree $y \in \mathcal{M}$ that captures the hierarchical entity structure in the mentions the best. In order to characterize the "goodness" of a tree, we treat it as a factor graph with two different types of potentials. Our factor graph equivalent of Figure 4-2 is shown in Figure 4-3

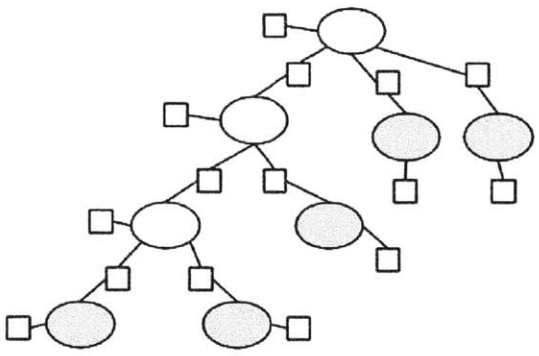


Figure 4-3: Factor graph representation of Figure 4-2. Shaded nodes correspond to the observed mentions, while unshaded nodes correspond to the latent sub-entity nodes.

Node potentials $\psi_i : V \mapsto \mathcal{R}$ measure the cohesiveness of a sub-entity. For example, a node potential might measure the Shannon entropy of the sub-entity's bag-of-

words.

Pairwise potentials $\psi_{ij} : V \times V \mapsto \mathcal{R}$ measure the similarity of two different sub-entities. For example, the potential $\psi_{ij}(v_i, v_j)$ might compute cosine similarity of the bag-of-words of v_i and v_j. In our work, we only consider using pairwise potentials between a sub-entity and its parent; implicitly this measures the similarity of a sub-entity with its siblings sub-entities. However, other pairwise interactions, such as between a node and one of its siblings, could similarly be defined.

Log-potential	Description
$\log \psi_{\text{bow}}(i,j) = \theta_{\text{bow}} \frac{i[\text{bow}] \cdot (j[\text{bow}] - i[\text{bow}])}{\|\|i[\text{bow}]\|\|\|\|j[\text{bow}] - i[\text{bow}]\|\|}$	cosine similarity of bag-of-words
$\log \psi_{\text{bon}}(i,j) = \theta_{\text{bon}} \frac{i[\text{bon}] \cdot (j[\text{bon}] - i[\text{bon}])}{\|\|i[\text{bon}]\|\|\|\|j[\text{bon}] - i[\text{bon}]\|\|}$	cosine similarity of bag-of-names
$\log \psi_{\text{bow}}(i) = -\theta_{\text{bow}} H(i[\text{bow}])$	entropy of bag-of-words

Table 4.1: Select potential functions. Recall that all edges (i, j) are tree edges; we assume implicitly that j is the parent of i. The notation $i[\text{bow}]$ means the bag-of-words corresponding to node i.

Table 4.1 shows some sample pairwise and node potential functions. For pairwise potentials between a node and its parent, we typically subtract the bags of the node from the parent in order to model the similarity of the child with its aggregate siblings (recall that we have constrained the state of each latent node to be the sum of its children). Note also that each potential has a corresponding constant θ, which serves to weigh one feature over others. For example, putting together the bag-of-words and bag-of-names features used in Table 4.1 we have

$$\begin{aligned}\log \psi_{ij}(y_i, y_j) &= \log \psi_{\text{bow}}(i,j) + \log \psi_{\text{bon}}(i,j) \\ &= \theta_{\text{bow}} \frac{i[\text{bow}] \cdot (j[\text{bow}] - i[\text{bow}])}{||i[\text{bow}]||||j[\text{bow}] - i[\text{bow}]||} + \theta_{\text{bon}} \frac{i[\text{bon}] \cdot (j[\text{bon}] - i[\text{bon}])}{||i[\text{bon}]||||j[\text{bon}] - i[\text{bon}]||}\end{aligned}$$

Thus each potential consists of the log-linear sum of potentials for each feature. It will be convenient to express this $\psi_{ij} = \exp\{\theta_E \cdot \phi(i,j)\}$ where

$$\begin{aligned}\theta_E &= [\theta_{\text{bow}}, \theta_{\text{bon}}]^T \\ \phi(i,j) &= \left[\frac{i[\text{bow}] \cdot (j[\text{bow}] - i[\text{bow}])}{||i[\text{bow}]||||j[\text{bow}] - i[\text{bow}]||}, \frac{i[\text{bon}] \cdot (j[\text{bon}] - i[\text{bon}])}{||i[\text{bon}]||||j[\text{bon}] - i[\text{bon}]||} \right]^T\end{aligned}$$

One might interpret the $\phi_{i,j}(y_i, y_j)$ vector as being a vector of statistics summarizing the pairwise similarities between the features vectors of node y_i and node y_j in the tree y.

We combined the factor potentials in a log-linear model. Thus the likelihood of the latent tree structure $y \in \mathcal{T}_{\mathcal{M}}$ is given as

$$P(y|\mathcal{M}) \propto \prod_{i \in V} \psi_i \prod_{(i,j) \in \mathrm{E}} \psi_{ij} = \exp\left\{ \left(\sum_{i \in V} \phi_i(y_i) \right) \cdot \theta_V + \left(\sum_{(i,j) \in E} \phi_{ij}(y_i, y_j) \right) \cdot \theta_E \right\} \tag{4.1}$$

where ϕ_is is a vector of statistics for single nodes, ϕ_{ij} is a vector of pairwise (usually between a node and its parents) statistics, and θ_V and θ_E represent the weights on these statistics.

4.2.3 Structural Regularization

From the form of Equation 4.1, one might rightfully suspect that our model favors tree structures with many edges (and therefore many sub-entity nodes, since $|E| = |V| + 1$ for a tree). Just as in other structure learning problems, invoking Occam's razor and selecting structures that are less complex might help in generalizing better. Our approach is to penalize certain structural statistics that are too high. There are several such structural statistics that we can consider. For example, we can penalize the number of nodes appearing in the structure.

Adding regularization to our model is simple. Let $S : \mathcal{T}_{\mathcal{M}} \mapsto \mathcal{R}^n$ be a function mapping a tree over the observed mentions to a vector of statistics about the tree structure. Table 4.2 shows some simple structural statistics. To penalize the statistic in our model, we can simply associate a regularization constant with each structural statistic. For example, to penalize the number of nodes in the structure, we include the term $-S_{NC} \sum_{y_i \in y} 1$, where the S_{NC} is a tuned regularization constant. We also note that many structural penalizations can be naturally interpreted as another sort of node potential (that is, the structural statistic can be computed by summing up some statistic at each node).

Structural Statistic	Description
$S_{NC}(y) = \sum_{y_i \in y} 1$	count number of tree nodes
$S_{LD}(y) = \sum_{y_i \in y} numleaves(y_i)$	depth of mention leaves
$S_C(y) = \sum_{y_i \in y} \frac{1}{\lvert numchildren(y_i) - k \rvert}$	absolute difference from having k children per node
$S_C(y) = \sum_{y_i \in y} \delta(a \leq numchildren(y_i) \leq b)$	number of nodes with $< a$ or $> b$ children

Table 4.2: Select list of possible tree structure statistics. We observe that many natural structure regularizers can be naturally be implemented as node potentials.

With this in mind, the regularized model becomes

$$P(y|\mathcal{M}, S) \propto \left\{ \left(\sum_{i \in V} \phi_i(y_i) \right) \cdot \theta_V + \left(\sum_{(i,j) \in E} \phi_{ij}(y_i, y_j) \right) \cdot \theta_E + S(y) \cdot \theta_S \right\} \quad (4.2)$$

where θ_S is the the vector of respective regularization constants for $S(y)$.

While in theory, we can combine many structural statistics into the vector $S(y)$, for this thesis we will only choose to penalize one regularization statistic at a time. This is to keep the model simple and make it easier to tune the regularization constants.

We will also note later that we will prefer structural statistics that are easily updated when local changes are made to the tree structure.

4.2.4 Entity Selection

As noted earlier, our approach divides the CDCR task into two parts:

1. (Tree Structure Learning) Infer latent tree structure over mentions, with hidden nodes representing some hierarchical notion of entities

2. (Entity Selection) Use the learned latent tree structure to select entities.

Most of the work of this thesis is dedicated to the first task. But once we have learned a tree structure over the mentions, selecting entities is equivalent to partitioning the tree into subtrees, such that each subtree root correspond to an entity. To accomplish this,

we can apply greedy clustering heuristic, similar to the methods used to find clusters in Hierarchical Agglomerative Clustering models [17, 12]. For example, we can use make use of simple compatability functions over nodes, measuring how cohesive its attributes are or how similar its children are. If the number of entities is roughly known in advance, then we can use this information too. In section 5.2, we describe our specific algorithm in some more detail

4.3 Comparision with Other Work

The most similar models to ours are by Sameer, Wick, et al. [22, 25].

The approach of Wick et al. [25] recursively decomposes each entity into sub-entity trees, in order to block Metropolis Hastings proposals. However, it does not model an hierarhcy over entities themselves. Sameer at al. [22] use "super entities" in order to efficiently distribute and parallelize inference with over similar entities. However this is only a one-level of entity hierarchy, which again serves only to encourage more successful Metropolis Hastings proposals. Wick et al. [25] does not model this "super entity" hierarchy, in favor of high-recall "canopy" functions to filter entities that are most likely to be corefererent.

Our work draws from these sub-entity concepts by allowing an arbitrary hierarchy over the entities as well. By just learning a tree structure over entities, one might expect entities that are more similar to be "closer" in the tree. In principle this will allow proposals to be kept local in the tree. Furthermore, allowing a full hierarchy may allow potentially interesting insights regarding the relationships between entities.

By expressing coreference explicitly in terms of structure learning, we are also able to naturally also formalize our notion of penalizing complex structures. Wick et al. [25] puts a -8 penalty on creating nodes in order to control tree depth. Our formulation allows us to formally think about these penalities as regularizing against trees with more complex structural statistics.

Chapter 5

Methodology

In this chapter we describe the implementations of the algorithms used to perform learning and inference for our model. As we described earlier, we draw inspiration from how a structure learning problem consists of determining three parts: the number of latent variables, the edges between the variables, the parameters describing the relationships between variables. Our inference procedure learns both the number of latent variables and the relationships (tree edges) between them, while our parameter estimation method will take care of the third part.

We first will describe the inference procedure that we use to select the "best" tree structure for a given model. Then, we describe how we use stochastic gradient descent-based estimation to learn parameters for the model.

5.1 Inference Method

We leave it up to the inference procedure to determine how many latent variables and which edges exist in the tree. This is not an easy problem; the number of partitions of n items is given by the Bell number $B(n) = O(n/\ln n)^n)$. Our hierarchical trees allow further sub-partitioning within each partition, thus the number of possible trees over n mentions is super-exponential in n, so it is intractable to consider all possible tree structures. To deal with this, we'll use Metropolis Hastings inference to efficiently explore the space of trees.

Let $P(y|\mathcal{M}, S)$ correspond to the regularized likelihood (Equation 4.2). To use Metropolis Hastings effictively, we must be able to efficiently compute the likelihood ratio $\frac{P(y'|\mathcal{M},S)}{P(y|\mathcal{M},S)}$. If we define our proposals to only affect a very local region of the whole tree, then many of the potentials will remain the same and cancel out in the likelihood ratio.

Also, because are interested in using Metropolis Hastings to perform MAP inference, we will use a temperature T very close to zero, which means that computing $q(\cdot|\cdot)$ in the acceptance ratio ratio is negible compared to the likelihood ratio, hence we ignore it. One might wonder if keeping T close to zero will cause inference to easily get stuck in local maxima. As we shall see in section 6.4, structural regularization helps inference stay away from these local maxima states.

5.1.1 Proposal Distribution

In this section we describe the proposal distribution that we use for Metropolis Hastings inference. We require a proposal distribution q that is ergodic and as local as possible.

Our proposal distribution first selects a random node and then randomly selects one of four actions. These actions and more actions are described in the technical report [21]. The actions correspond to: create/deleting sub-entity nodes and raising/lowering sub-entities.

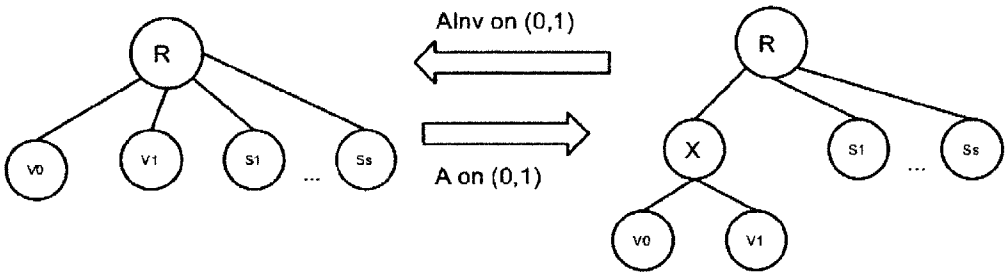
1. A (Sub-entity Create): Take two sub-entity siblings and create a new sub-entity with them as children.

2. A^{-1} (Sub-entity Delete): Destroy a sub-entity (if it has only two children) and join its children with the parent sub-entity.

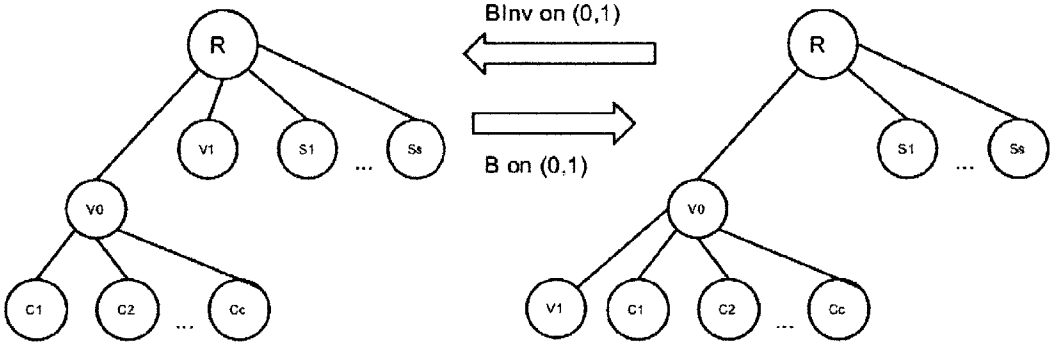
3. B (Sub-entity Up): Join a sub-entity with one of its siblings, pushing it down a level.

4. B^{-1} (Sub-entity Down): Join a sub-entity with its grandparent, pulling it up a level.

These proposals are visualized in Figure 5-1a and Figure 5-1b. The A, A^{-1} pair is responsible for creating or destroying sub-entities, while the B, B^{-1} pair is responsible for moving sub-entities without creating or destroying any nodes. Recall also that after each proposal, we enforce that the bags of a node equal the aggregate bags of its children.



(a) Action A and A^{-1}: these moves create or destroy sub-entities by moving the sub-entities labeled v_0, v_1. The node labeled X is the created/destroyed sub-entity.



(b) Action B and B^{-1}: these moves move sub-entities up or down a level in the tree by moving the sub-entity labeled v_1.

Figure 5-1: Actions for our proposal distribution.

In general, the number of potentials that each proposal has to consider is at most linear in the number of the selected node's children. Appendix A shows the computations needed to compute the likelihood ratio exactly. It is clear that the set of actions we have defined is sufficient for exploring the full state space of trees with the constraint that each non-leaf has at least one child.

5.1.2 Full Inference Algorithm

The full inference algorithm is described in Figure 5-2. We always begin with an initial state where all the observed mention nodes are children of a root node. Thus in this state, if we were to declare each child of the root as an entity, we would obtain perfect precision (since each hypothesized entity contains exactly one mention) but a poor recall. Thus the inference procedure will first combine mentions that seem sufficiently similiar (via the model $P(\cdot|\mathcal{M}, S)$) into subtrees, and then be able to combine/split and move these subtrees up/down.

1. Initialize with a tree y with no sub-entities, where each mention is a child of a root node.
2. For $i = 1, 2, \ldots$
   (a) Select a tree node uniformly at random
   (b) Attempt one of the local actions $\{A, A^{-1}, B, B^{-1}\}$ uniformly at random, conditioned on the proposal being legal with respect to the structure constaint (each non-leaf has at least one child) to get new tree y'
   (c) If Metropolis Hastings accepts y'
      i. Update $y = y'$

Figure 5-2: Full Inference Algorithm

Similar to other work using Metropolis Hastings for coreference [22, 25], the MCMC sampling can be made massively parallel. The proposals in a subtree s are not affected by proposals that occur outside the in the Markov blanket (treating out factor graph like an undirected graphical model) of s. Although parallelization strategies and related infrastructure are not the focus of this thesis, in section 7.2 we will briefly discuss what is possible.

5.2 Entity Selection

As mentioned previously, we can use a variety of different greedy clustering heuristics to partition the tree structure obtained from inference into subtrees corresponding to

entities. The approach we take is shown in Figure 5-3.

1. Initialize min priority queue with the root node
2. While termination criterion is not met:
   (a) Pop node n from priority queue.
   (b) For each of its children $n.c_i$:
      i. Compute a "goodness" score for $n.c_i$ and enqueue using the score as priority
3. Return nodes in priority queue

Figure 5-3: Greedy Clustering Heuristic

There are several choices of "goodness" score. Other possible choices in include a weighted Shannon entropy of features. For termination criterion, one might use a tuned threshold or the number of entities, if known. For this thesis we use the average log-potential $\frac{1}{|children(n)|}\sum_{c_i \in children(n)} \log \psi(n, c_i)$ and the correct number of entities as the terminal condition. More details about the specific clustering heuristic we use appear in [21].

One might be suspicious about us using the correct number of entities to perform entity selection. Several other works [1, 22, 25] use tuned parameters in order for their techniques to obtain the correct number of entities. We could very well do the same and learn a termination criterion that generalizes well. However, we would like to to point out that the exact entity selection heuristic we use is not central to this thesis. We provide a method for entity selection in order to demonstrate that it is possible to use the tree structure learned from our model to easily obtain potential entities for CDCR. This demonstrates that the latent tree structure that we learn from our approach is meaningful.

5.3 Parameter Estimation

We would like to learn parameters that allow the inference procedure to find tree structure that captures the latent entity structure in the mentions. Given a labeled sample of mentions, we can estimate parameters via stochastic gradient-based methods. Figure 5-4 shows the general approach. We note that the gradient ∇ is computed using the difference in statistic vectors ϕ (as defined in Equation 4.1).

Note that for this parameter estimation step, we ignore the structural regularizations; we will discuss how to tune these in the next section.

1. Initialize $\theta^{(0)} = 0$
2. Create tree y such that there is exactly one sub-entity for each unique entity.
3. For $i = 1, 2, \ldots$
   (a) Select entities e_1, e_2. Obtain new tree y' by proposing either:
      i. Move entity e_1 into e_2 (Sub-entity Move Down)
      ii. Moving mention $m_1 \in e_1$ from e_1 to e_2
   (b) Compute the gradient
   $$\nabla = \langle \phi_{v_1}, \ldots, \phi_{V_{|V|}}, \phi_{e_1}, \ldots \phi_{e_{|E|}} \rangle_{y'} - \langle \phi_{v_1}, \ldots, \phi_{V_{|V|}}, \phi_{e_1}, \ldots \phi_{e_{|E|}} \rangle_{y})$$
   from changing state from y to y'
   (c) If certain conditions hold, update
   $$\theta^{(i+1)} = \theta^{(i)} + \eta \nabla E$$

Figure 5-4: Parameter Estimation Method

In particular, we use the Contrastive Divergence approach of making a single Metropolis Hastings step away from some ground truth state, which will give us biased but low-variance estimates of parameters. The ground truth state is taken to be a tree where all the mentions in a subtree belong to the same entity.

If we always update the parameters using the gradient in step 3(c), then the approach is the same as Contrastive Divergence. The "certain condition" in step 3(c) gives us the flexiblity to make smarter parameter updates.

For example, if we had an objective function $F(\cdot)$ and update parameters only if $\theta \cdot \nabla < F(y^+) - F(y^-)$, as in Figure 3-2, then we obtain a variant of SampleRank in which we obtain constraint violations by considering only pairs of adjacent states that include the ground truth state. Such a methodology gives us a biased version of the SampleRank algorithm, but in return we get low-variance parameter estimates.

5.3.1 SampleRank Objectives

As descibed in the previous section, we can use an objective function to get SampleRank-like updates to our parameters. A natural choice of such an objective function is the B-CUBED F_1 measure (Figure 2-2), since this is the metric we will use to evaluate our coreference results. However, computing the changes in the the B-CUBED F_1 measure still requires us to maintain all of the hypothesized and true entity clusters. So if there are K unique entities, we are going to need $\Omega(K)$ extra space in order to compute the B-CUBED F_1 score for the new state, which maybe problematic if we have, for example, a million entities.

On the other hand, the pairwise F_1 measure (Figure 2-1) can be updated using only $O(1)$ extra space. The full details behind how appear in Appendix B. While scaling our approach is not the focus of the thesis, we will investigate how much parameter estimation performance varies if we choose to use B-CUBED or pairwise F_1 score as the objective function.

5.4 Structure Regularization Tuning

After computing the likelihood parameters θ, the next step is to tune structural regularization constants. A simple grid search is hard to generalize, because the scale of the regularization constants seems to very greatly depending on the particular corpus and regularization statistic.

For this thesis, we only explore the effects of using one regularization statistic during inference. If the B-CUBED F_1 performance of a learned tree stucture is approximately a unimodal function of the respective regularization constant, then we

can make use of standard search techniques, such as ternary search.

Figure 5-5, shows how we perform regularization constant tuning using ternary search. Before running the ternary search, we first normalize the likelihood parameters θ so that $\sum_i \theta_i = 1$, so that we can be confident that the regularization constant lies in the interval $[0, 1]$.

1. Initialize $l = 0, r = 1$
2. While $|l - r| > \epsilon$
   (a) Compute $lt = \frac{2l + r}{3}, rt = \frac{l + 2r}{3}$
   (b) Run MCMC inference using lt, rt as regularization constants to get respective F_1 scores F_{lt}, F_{rt}.
      i. If $F_{lt} < F_{rt}$ then assign $l = lt$
      ii. Else assign $r = rt$
3. Set $S^* = \frac{l + r}{2}$ as regularization constant

Figure 5-5: Ternary Search for Regularization Constant Tuning

For a given tolerance ϵ, the total number of calls to MCMC with this approach is $\Theta(2 \log_{2/3} \epsilon)$. For example, if we require a tolerance of $\epsilon = 10^{-3}$, then we need $2 \log_{2/3} 10^{-3} = 34$ calls to the MCMC procedure. Each MCMC procedure takes a non-neglible amount of time, making this regularization tuning method slow compared to the other methods. Since most of the tuning time is dominated by the time to perform MCMC inference, it is possible to use smarter search techniques such as Fibonacci or golden section search [10], which can reduce the number of calls to MCMC by 50%. However for this thesis, we use stick with the naive ternary search.

5.4.1 Generalizing Regularization Constants

One final consideration has to be made when considering how to properly tune the regularization constants. Suppose that the corpus used to tune the regularization constants is size n_T, but the corpus for validation (or new data) is of size n_V. Fur-

thermore suppose that $n_T \ll n_V$ or $n_T \gg n_V$. Then one might expect the magntiude of the structural statistics for the best trees for the two corpora to be quite different. Does this affect the regularization constants?

Proposal Action	$S_{NC}(y') - S_{NC}(y)$	$S_{LD}(y') - S_{LD}(y)$
A	$+1$	+ (num leaves in both subtrees)
A^{-1}	-1	$-$(num leaves in both subtrees)
B	0	+ (num leaves in subtree)
B^{-1}	0	$-$ (num leaves in subtree)

Table 5.1: Changes in the NC (number of nodes) and LD (sum of depth of the leaves) structural statistics, for each of the four Metropolis Hastings proposal actions.

Table 5.1 shows that sometimes the size difference might be important. For each of the the four proposal actions that can be made during MCMC inference, the NC (number of nodes) structural statistic changes by a constant. However, the LD (sum of depth of leaves) statistic changes in magnitude proportional to the size (in number of leaves) of the sub-entities. Because this average size scales with the size of the corpus, we expect that we will have to make a correction for this regularization constant when we generalize to a differently-sized corpus. For example, if the training corpus is of size $n_T = 100$ and the validition corpus is of size $n_V = 300$, we expect on average for the subtrees of the validation corpus to be three times as large as in the training corpus. If we were to use the same regularization constant S_{LD} for both, then proposals in the validation corpus would on average be penalized three times as much.

To resolve this problem, we define the *size correction factor* to be the ratio $\frac{n_T}{n_V}$. For the LD statistic, if $S^*_{LD}(n_T)$ is the tuned regularization constant for the training corpus, then for a corpus of size n_V, we adjust

$$S^*_{LD}(n_V) = \frac{n_T}{n_V} \cdot S^*_{LD}(n_T)$$

For example, if $n_T = 300, n_V = 100$ then $S^*_{LD}(n_V) = 3S^*_{LD}(n_T)$. This means for smaller corpus the regularization constant should be larger, and for a larger corpus the regularization constant should be smaller.

We note that not all structural statistics need a size correction to generalize (for example, the node count statistic), and different statistics might need to be generalized differently. This section demonstrates that such considerations, although need not be complex, must be taken into consideration.

5.5 Full Learning Algorithm

We will now summarize our complete learning algorithm, which is shown in Figure 5-6. We first use our stochastic gradient descent-based learning method to obtain likelihood model parameters, which are appropriately normalized. Then a regularization statistic is tuned using ternary search and possibly transformed in order to generalize.

1. Likelihood Parameter Learning
   (a) From labeled training set perform stochastic gradient descent-based learning (e.g. SampleRank or Contrastive Divergence) to obtain likelihood model parameters θ
   (b) Normalize $\sum_i \theta_i = 1$.
2. Regularization Constant Tuning
   (a) Pick structural statistic $S_x(\cdot)$
   (b) Perform ternary search on regularization parameter S_x
   (c) Consider applying a size correction on S_x or another transformation, depending the particular regularization structural statistic chosen.

Figure 5-6: Full Learning Algorithm

Both the likelihood parameter learning and regularization constant tuning steps are parallelizable. In section 7.2 we will discuss how to potentially scale these procedures.

Chapter 6

Results and Analysis

6.1 Datasets

We use two corpora to evaluate our methods and compare them with previous approaches.

First, to compare with related work, we use the John Smith (JS) corpus, which consists of 197 articles from the 1996 and 1997 editions of the New York Times, labeled to obtain 35 true entities. The only criterion for including an article was that the article included a string matching the regular expression `John.*?Smith` [1]. Because the data has practically no name variation, this is strictly a disambiguation task.

Our second corpus is a sample from the Wikpiedia Links corpus (WL) [22]. The WL corpus treats Wikipedia pages as entities and links to different pages as being mentions of the respective entities. As the original WL corpus consists of over a million mentions, we sample a subset of 243 documents that contain a mention of the name 'Eric' (henceforce, the WLE corpus), with a total of 57 unique entities. Since we only filtered by first name, unlike in the JS corpus, most of the entities have unique last names. Thus, the WLE corpus can be thought of the opposite of the JS corpus. Because names have strong predictive power, we look for our learning procedure to be able to discover this.

6.2 Parameter Estimation Comparisions

To test our parameter estimation method, we use only three features: bag-of-names (bon), bag-of-words (bow), and entity-type (type). These features were obtained using the BBN SERIF [2] system. The bag-of-names feature is sufficient to deal with name variation (which is helpful for the WLE corpus but not the JS corpus). The bag-of-words feature is useful for name disambiguation. The final feature, entity-type, is the same among all the mentions (person). The purpose of including this feature is to add noise to the feature set (since the cosine similarity between two sub-entities with this feature will always be 1). Thus, this helps us test how well our parameter estimation method is able to distinguish signal from noise.

We first evaluate our parameter estimation framework by comparing the estimates of Contrastive Divergence and SampleRank for the three features on the JS our WLE corpus. Table 6.1 compares the parameter estimates. Note that we trained SampleRank using both the Pairwise F_1 and B-CUBED F_1 metrics as objective functions.

Training Algorithm	θ_{bow}	θ_{bon}	θ_{etype}
Contrastive Divergence	0.4689	0.2796	0.2514
SampleRank w/ Pairwise F_1	0.9318	0.06350	0.004698
SampleRank w/ B-CUBED F_1	0.8789	0.05494	0.06618

(a) John Smith corpus

Training Algorithm	θ_{bow}	θ_{bon}	θ_{etype}
Contrastive Divergence	0.3827	0.5011	0.1162
SampleRank w/ Pairwise F_1	0.3909	0.6082	0.0008986
SampleRank w/ B-CUBED F_1	0.3661	0.6117	0.02212

(b) Wikilinks Eric corpus

Table 6.1: Parameter Estimates, where the three available features are bag-of-words (bow), bag-of-names (bon), and entity-type (etype).

We see that for both corpora, Contrastive Divergence does a poor job of realizing that the entity-type feature is noise, giving it 25% of the weight in the JS corpus and 11% of the weight in the WLE corpus. The estimates for SampleRank using the Pairwise F_1 measure and B-CUBED F_1 are quite similar. For the JS corpus, both are able to detect that the bag-of-words is the strongest feature, giving it roughly 90%

weight in both cases. For the WLE corpus, both give bag-of-names (the strongest feature) about a 60% weighting. Thus, from these two corpora, it seems that we might well be able touse Pairwise F_1 (which can be updated more efficiently) as a proxy for B-CUBED F_1.

6.3 Local Maxima States During Inference

In this section we describe the common class of local maxima that our model enters during MCMC inference. To do this, we look at two runs of MCMC inference on the exact same sample of 74 mentions from the WLE corpus. This corpus sample was selected because the following results were easily reproducible using it.

Model parameters were learned using SampleRank with Pairwise F_1 objective on the whole WLE corpus (note that we do not need to worry about overfitting, since we are just studying the behavior of the actual inference procedure).

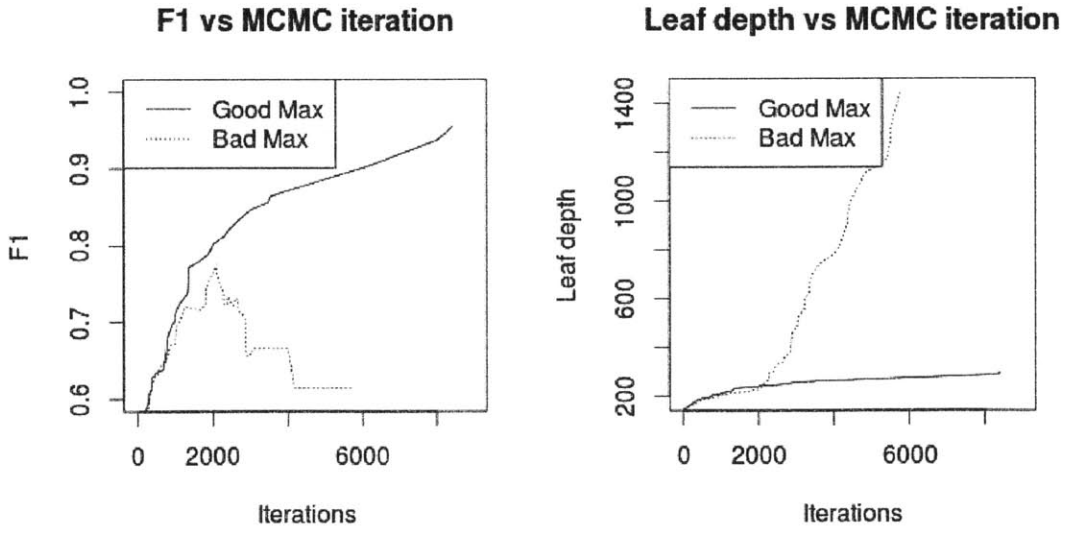


Figure 6-1: Performance of two MCMC runs for the exact same corpus and model parameters. One ends up at a good local max and other at a bad local max, as indicated by the F_1 score. We can see that the structure of the bad run explodes in complexity at the same time.

As the left plot in Figure 6-1 shows, in one run MCMC inference is able to find

a tree structure that has F_1 score close to 1 (labeled: "Good Max"). In the other run (labeled: "Bad Max"), after 2,000 Metropolis Hastings iterations, the inference procedure begins entering a local maximum state as the F_1 plummets from over 0.75 to 0.60 in the next 4000 iterations.

We can see what is happening structurally in the right plot of Figure 6-1. After 2,000 iterations, the bad run starts to explode exponentially in the sum of leaf depth, which corresponds to a deep chain being built during inference. Thus the local maximum is a result of an extremely complicated structure being formed.

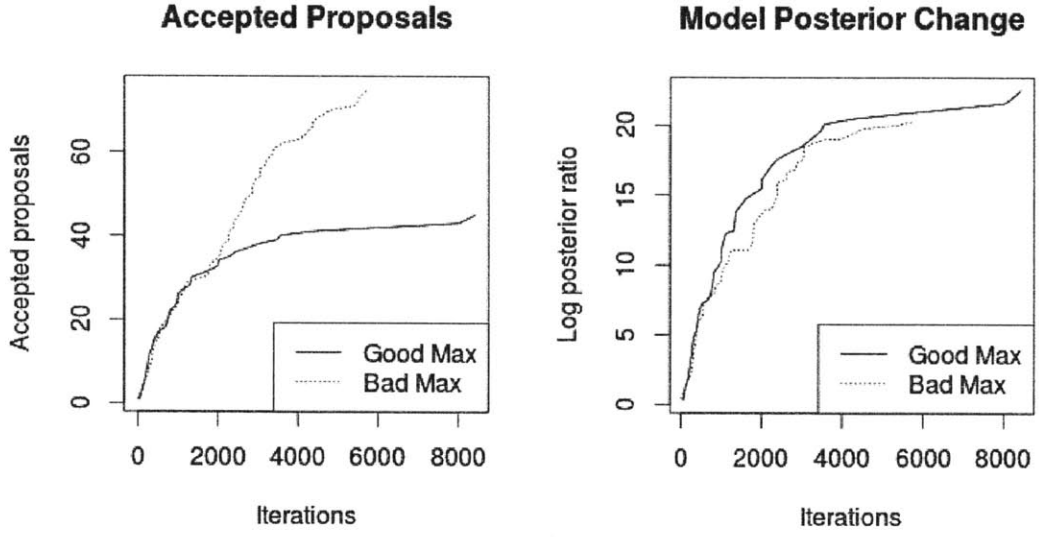


Figure 6-2: Comparision of Metropolis Hastings performance for the same "good" and "bad" MCMC runs as in Figure 6-1. The number of accepted proposals and change in the model log-posterior are shown.

To get another perspective, Figure 6-2 shows what is happening with the Metropolis Hastings proposals themselves. The nature of the problem is clear: in the bad run, Metropolis Hastings accepts several proposals which only increase the model score slightly. A likely explanation is that just due to just random noise in the features, it's possible for a small positive log-likelihood ratio to result from joining two non-similar sub-entities.

Figures 6-1 and Figures 6-2 show that once bad proposals start to be accepted,

more are accepted and leaf depth grows exponentially as the deep branch gets deeper; there seems to be a snowball effect. Qualitatively, we observed that this is because when the number of children of the root is small, the average likelihood ratio for *any* proposal increases slightly. Furthermore, once sub-entities of a particular entity are absorbed into the long branch, its other sub-entities will have more inclination to join it.

Since the problem stems from many proposals with very neglible likelihood ratio being accepted, one way is fight against this is to bias the log acceptance ratio to be

$$\log \alpha = \min \left(0, \log \left(\frac{P(s')}{P(s)}\right) - b\right)$$

However it seems hard to pick a good bias b and such a bias is not very interpretable.

Structural regularization is able to accomplish the exact same goal and can be be interpreted. In the next section, we will discuss the effects structural regularization has on inference.

6.4 Structure Regularizers

As we have seen, structural regularization seems to be a necessary feature in our model, in order to generalize well.

In this thesis, we consider two natural regularization statistics: the tree node count (NC) and the sum of the depth of the leaves (LD), as defined in Table 4.2. We pick these two because they are natural measures of structural complexity. A tree can only be complex if it has many nodes, and if the sum of the depth of leaves is high, then a long branch in the tree is likely present. It is also clear that these two structural statistics are correlated.

In this section we answer two question:

1. For a given corpus, how does changing the regularization statistic and corresponding constant affect inference?

2. How well does a regularization constant generalize, when we increase the size of

the corpus? For example, if the regularization constant works well for a random sample of the corpus, how well will it work well for the whole corpus?

For these experiments, we use the likelihood parameters that are learned using SampleRank with Pairwise F_1 on both the JS and WLE corpus. With these paramters fixed, we now analyze the effects that structural regularization has on inference.

6.4.1 Relation with Regularization Constant

We first analyze the effects the regularization constant has on inference. There are two cases we consider: JS with NC, JS with LD. For each case, we swept over regularization constant values in $[0, 1]$, running MCMC inference for at most 80,000 iterations.

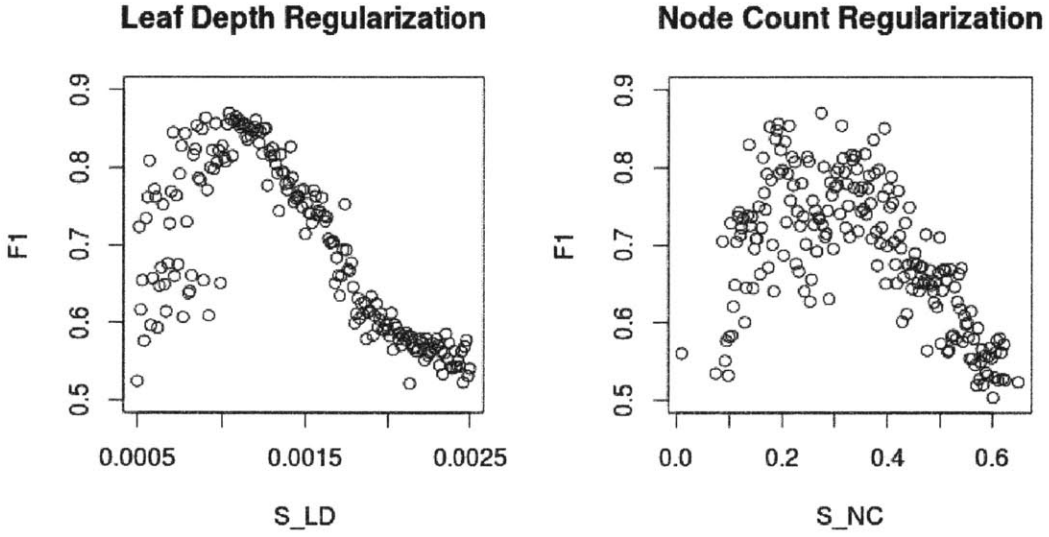


Figure 6-3: JS corpus, F_1 score of tree learned from MCMC inference as a function of regularization constant. For both NC and LD structure statistics, F_1 as a function of the regularization constant seems to approximately unimodal.

In Figure 6-3 we see that both the Leaf Depth (LD) regularization and Node Count (NC) regularization regimes are approximately unimodal. This justifies the use of a ternary search to tune the regularization constant.

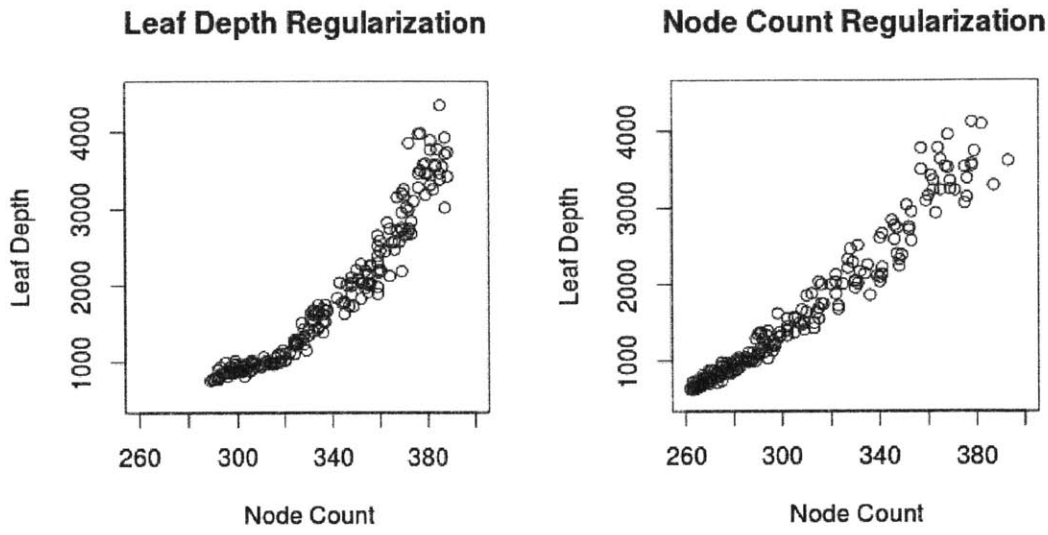


Figure 6-4: JS corpus, leaf depth as a function of node count for both regularization both the LD and NC structural statistic.

We also observe that it appears that LD regularization seems to have less variance than the NC regularization. One explanation is that forcing a tree to have a small LD is stronger than forcing it to have a small NC; two trees with the same NC can have very different LD. In Figure 6-4 we see that for the exact same data, under NC regularization LD seems to grow linearly with NC. However under LD regularization, the growth is sub-linear. So in this sense, LD penalization is stronger than NC penalization.

6.4.2 Relation with Corpus Sample Size

We next investigate how well the regularization constants generalize, in the sense that a structure regularization tuned on a sample works well with the original corpus. Our strategy is to take different-sized samples from a corpus, and study how well inference performs as a function of the size of the sample. The methodology for doing this is summarized in Figure 6-5. The initial constant c is picked arbitrarily.

We repeated this experiment for four case: JS with LD, WLE with LD, JS with NC, and WLE with NC. The results, shown in Figure 6-6, are very interesting. First,

1. Let N is the size of the original corpus, c be an arbtirary regularization constant for the particular structural statistic
2. For structural regularization constant $S \in \{1/2c, c, 2c\}$:
   (a) For $n = 2, \ldots, N$:
      i. Sample the first n mentions from corpus
      ii. Run MCMC inference, using S as the regularization constant

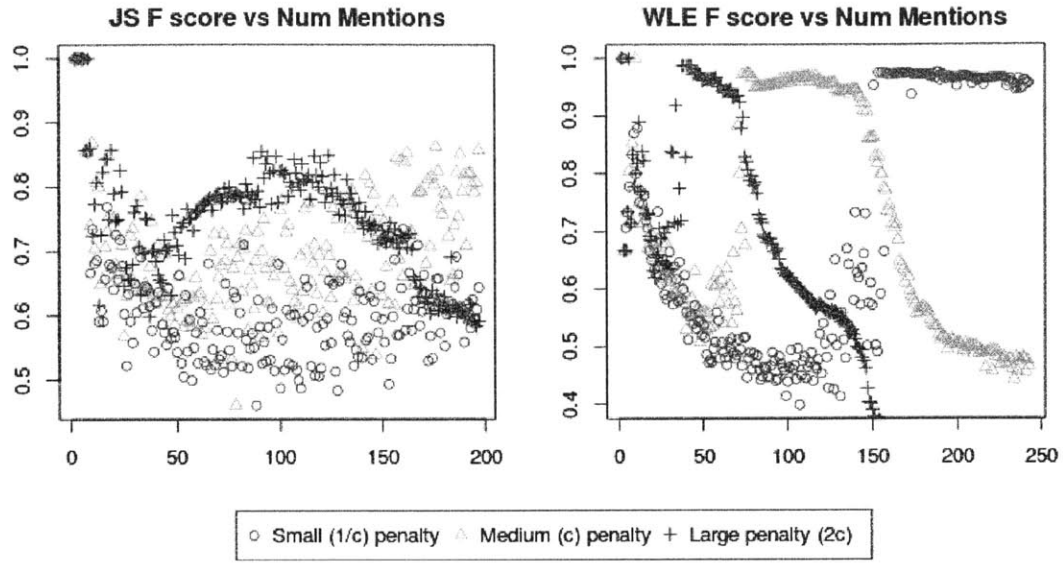
Figure 6-5: Procedure for assessing effects of corpus size on the effectiveness of a structural regularization constant.

as we might expect, inference on the JS corpus is noisier than on the WLE corpus (since the latter has bag-of-names as a high-signal feature), but the general trends are similar. As we suspected, under LD regularization, particular regularization penalties S_{LD} are effective for different sample sizes. In fact, there seem to be sharp "phase transitions" in the quality of inference, which can be seen in the right image of Figure 6-6a. We will explain these phase transitions in the next section.

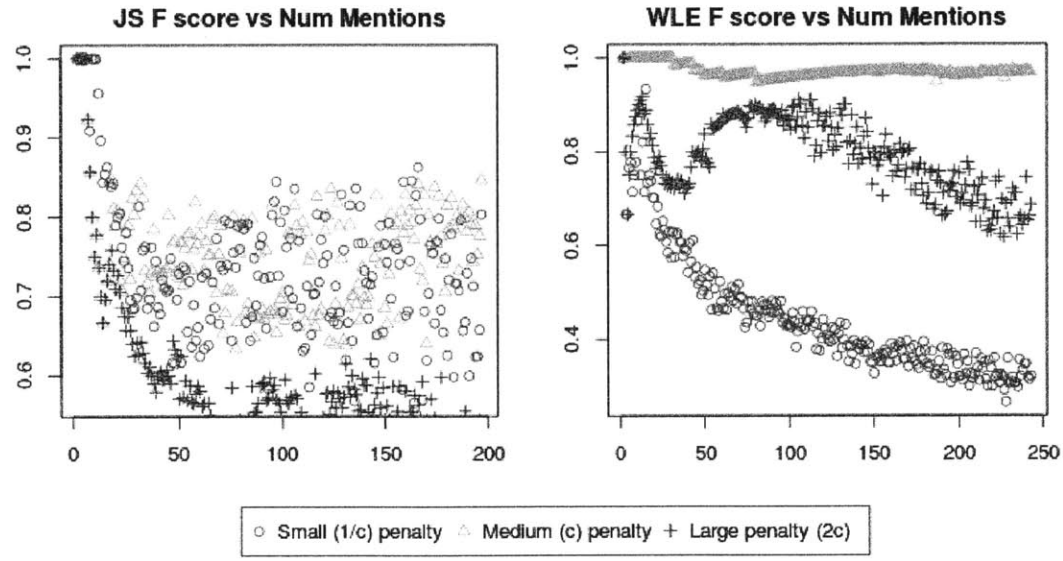
The right plot in Figure 6-6b seems to indicate that F_1 score sometimes decreases as we increase the corpus size, but this is not due to the regularization constant being less effective for different sample sizes. For example, for the WLE corpus both of the trees whe $n = 75$ and $n = 150$ appear to be random trees, so the decreasing F_1 score likely is just an asymptotic F_1 score for random trees. Thus unlike LD regularization, the NC regularization seems to be invariant to corpus sample size, which is what we suspected.

6.4.2.1 Phase Transitions for Leaf Depth Regularization

We now explain the phase transitions that appear in the right plot of Figure 6-6a. In particular, when penalizing the LD structure statistic, for a fixed regularization constant S_{LD}, there seems to be two critical values $\alpha^-(S_{LD})$ and $\alpha^+(S_{LD})$ of sizes of corpuses n for which the F score changes dramatically. The corresponding regions are:



(a) Leaf Depth Regularization



(b) Node Count Regularization

Figure 6-6: Comparision of when particular structural statistics and corresponding regularization constant work best during MCMC inference, tested on both the JS and WLE corpus. The regularization penalty c is chosen arbitrarily. The results indicate that while NC regularization seems mostly invariant to corpus size, LD has sharp phase transitions in the performance of MCMC inference, depending on the corpus sample size.

1. $n > \alpha^+(S_{LD})$ ("Right" region): Since all mentions are initialized as children of the root (as described in Figure 5-2), if an entity with $m = m_1 + m_2$ mentions is split into subtrees of size m_1 and m_2, the only way to combine them into one entity is to increase the leaf depth by m by moving both under same subtree. The sharp decline in F_1 score occurs when the largest entities are not able to be combined, hurting recall score dramatically.

2. $n < \alpha^-(S_{LD})$: ("Left" region): MCMC inference is prone to entering a local maxima state of creating extremely deep trees by arbitrarily combining subtrees, hurting both precision and recall dramatically.

3. $\alpha^-(S_{LD}) < n < \alpha^+(S_{LD})$ ("Middle" region): Ideal corpus sample size for S_{LD}. MCMC is able to join sub-entities together (avoiding the "Right" region), and there is sufficient regularization so that it is unlikely to enter local maxima state ("Left region")

6.4.2.2 Size Correction Empirical Justification

Previously we postulated that a size correction factor might be needed in order to generalize the regularization constant learned when using LD as the structural statistic. To empirically verify this correction, we compare at the median value of sample size for which each penalty is best. In particular, for each corpus and penalization contant, we filter for all trials where the F_1 score is at least 90% of the maximum achieved F_1 score for the corpus, and then take the median sample size for these points. The results are shown in Table 6.2. Note that as Figure 6-6a shows, the $1/2c$ penalty does not have any trials greater than the threshold.

Corpus	1/2c penalty	c penalty	2c penalty
JS	NA	173	96
WLE	199	111	55

Table 6.2: Median of ideal corpus sample size ranges, for each of the three values of the LD regularization constant.

Our size correction factor approximately matches Table 6.2. For example in the

JS corpus, we observe that $173/96 = 1.80 \approx 2$. For the WLE corpus, we see that $199/111 = 1.79 \approx 2, 111/55 = 2.02 \approx 2$, and $199/55 = 3.62 \approx 4$.

6.4.3 Comparision of Regularization Statistics

We now compare MCMC inference across our three structural regularization regimes:

1. No structure regularization
2. Sum of Leaf Depth (LD) regularization
3. Node Count (NC) regularization

For these experiments, we use our parameter estimation and regularization tuning method to train on half of the John Smith corpus. The behaviors of MCMC inference on the held-out half are shown in the proceeding experiments.

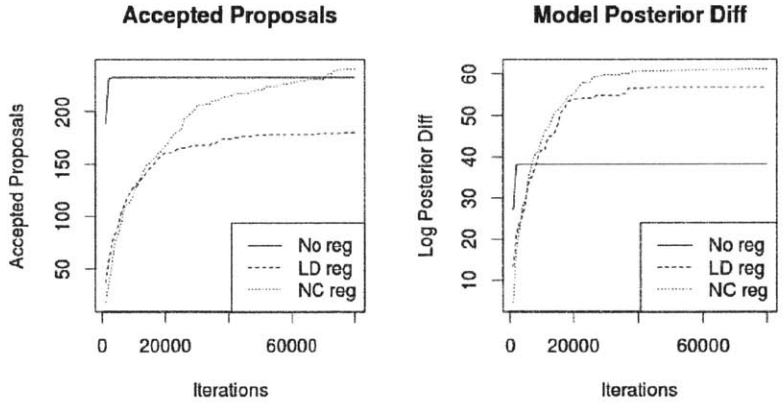


Figure 6-7: Comparision of MCMC proposal acceptances and corresponding change in log-posterior using no regularization, LD regularization, and NC regularization on the JS corpus.

First, we look at how many Metropolis Hastings proposals are accepted in each of the three regimes, and the corresponding changes in log model score . Figure 6-7 shows that having no regularization accepts many proposals early in the inference procedure and then gets stuck in local maximum, as described in section 6.3. Both two regularization regimes accept proposals much more slowly. Between LD and NC

regularization, although LD and NC regularization, it seems that NC regularization will accept more proposals than LD regularization that increase the model posterior only slightly (as indicated by the right plot).

Figure 6-8 shows what is happening structurally during MCMC inference. First without regularization, $F1$ score increases very quickly after a few thousand iterations but reaches a local maximum around 0.45, whereas using NC and LD regularization, the F_1 score increase more slowly initially but reach arond 0.80 F score.

Both LD and NC regularization have slower growth and overall number of nodes than the unregularized case. It seems that directly penalizing the number of nodes results in substantially fewer nodes when we do NC regularization. Interestingly in these runs of MCMC, LD regularization approaches a LD statistic similar to the unregularized case, but without hurting the F_1 score. Overall, it seems that structural regularization is doing a good job of preventing bad proposals with small change in model score from being accepted, while controlling the growth of complexity in the tree structure.

6.5 Entity Hierarchy Evaluations

In this section, we analyze whether if the latent hierarchy over entities is meaningful. In particular, one might might suspect that a potential advantange of allowing such a hierarchy is that entities that are most similar will be close in the tree structure. Proximity in the tree facilitates proposals between entities that are similar.

To quantify this, we consider a tree learned from MCMC inference, using model parameters and regularization constants trained from our learning method in Figure 5-6 and SampleRank w/ Pairwise F_1 objective. A natural notion of proximity in a tree is least-common ancestor (LCA) distance. Our hypothesis is that entities that have high similarity have smaller LCA distance in the tree. Similarity is computed as the weighted sum of cosine similarities of features, using the estimated model parameters. Figure 6-9 shows our methodology for testing our hypothesis.

For each pair of entities that we identify in the tree, we consider their LCA distance

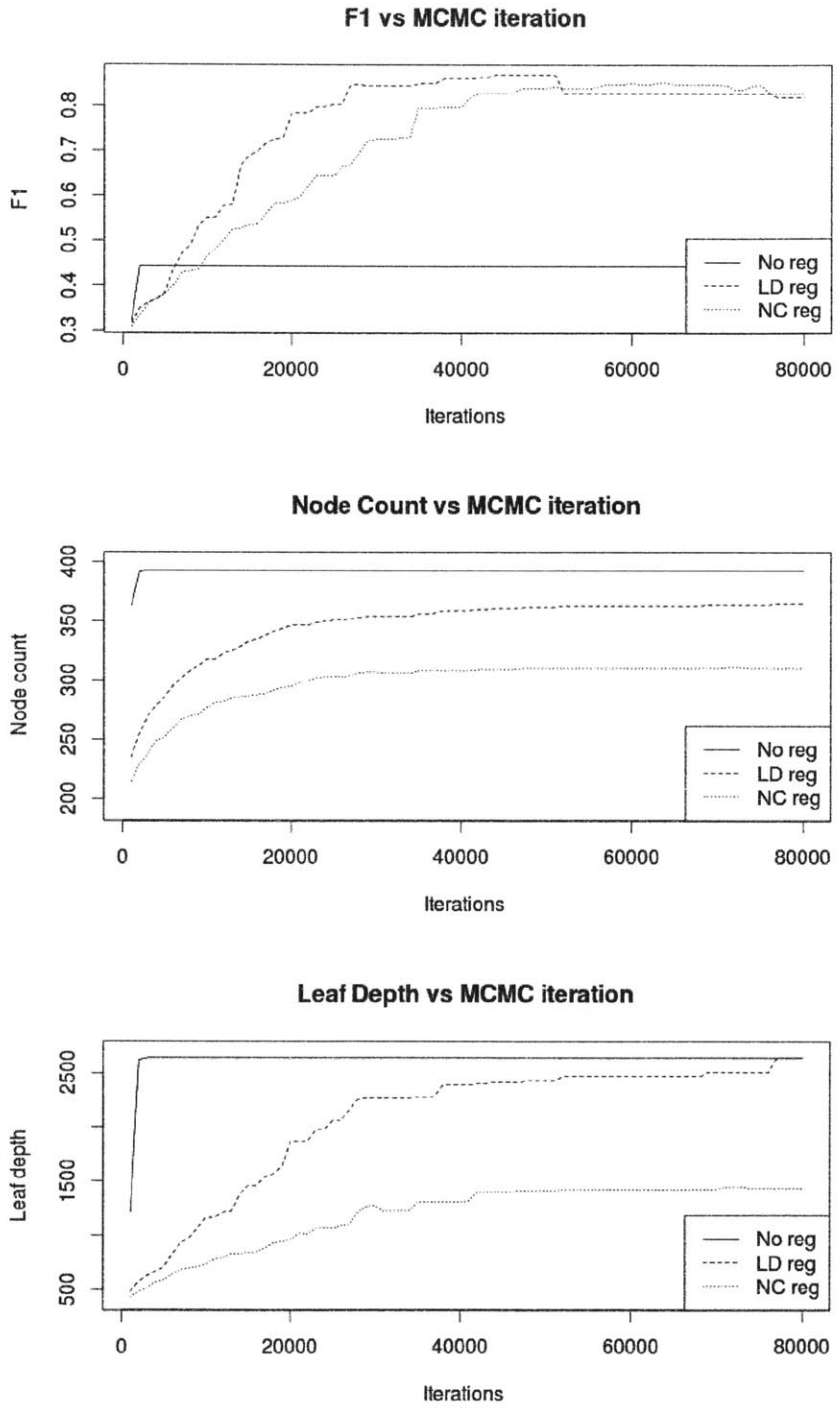


Figure 6-8: Comparision of MCMC inference using the three different regularization regimes: how F_1 score and structural statistics evolves over time.

1. Using model parameters and regularization constants trained from learning method, perform MCMC inference.
2. Apply greedy clustering herurstic to obtain set of entities $\mathcal{E}$.
3. For all unique pairs of entities $e_i, e_j \in \mathcal{E}$. With $|\mathcal{E}|$ entities there are $|\mathcal{E}|(|\mathcal{E}|-1)/2$ such pairs:
   (a) Find the lowest common ancestor of e_i and e_j in the tree. Let d_{ij} be the sum of the distance from e_i to the LCA and the distance from e_j to the LCA.
   (b) Let s_{ij} be the pairwise similarity between e_i, e_j computed using the model parameters.
4. Finally run a regression of d on s.

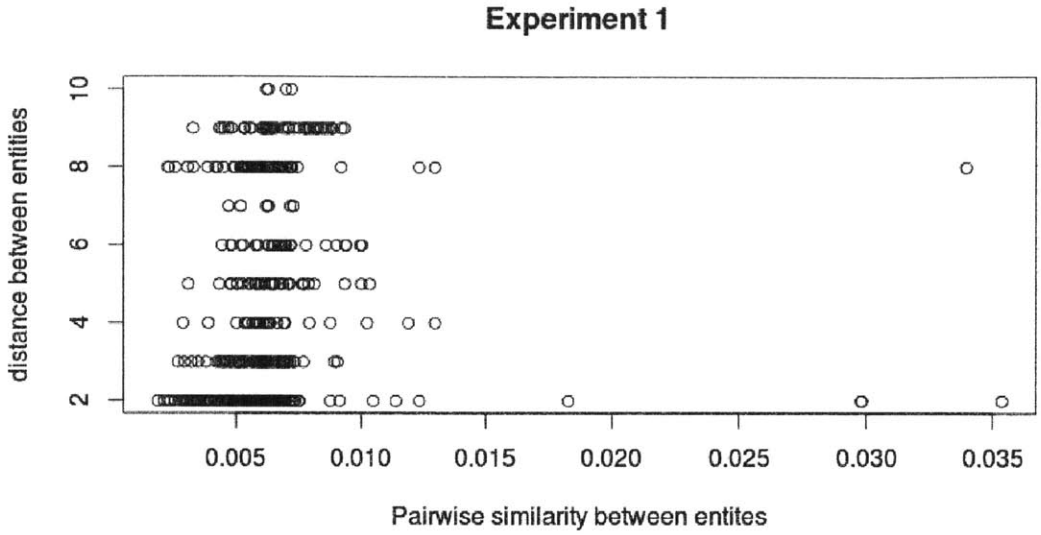
Figure 6-9: Method for Evaluating Latent Hierarchy of Entities

and their simliarity (measured via the pairwise potentials in our likelihood model). After obtaining these, we ran a regression of thee LCA distance on simlarity.

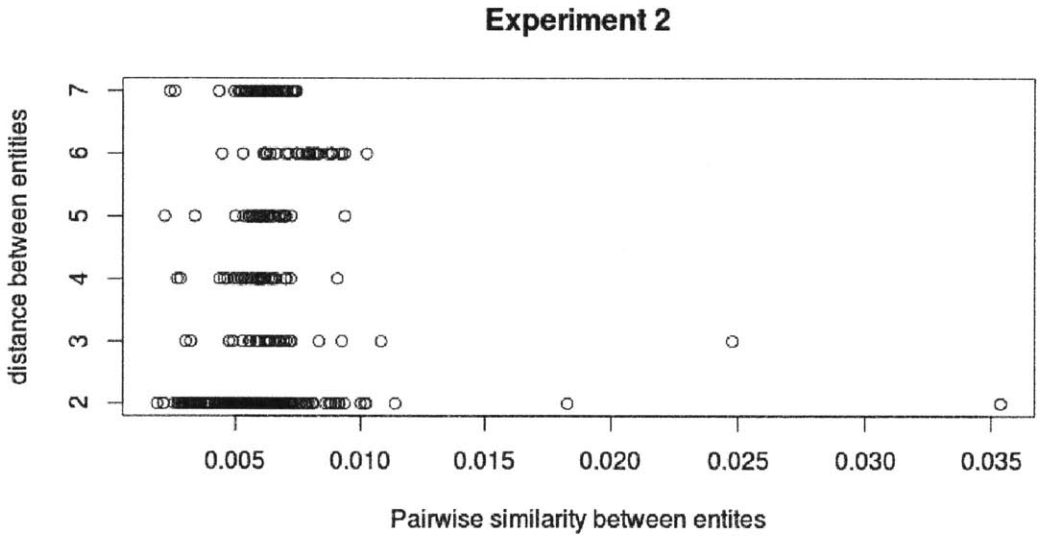
	Regression Coefficient	p-value
LD Regularization	89.5546	0.0109
NC Regularization	119.4714	0.00303

Table 6.3: Results of performing regression of LCA distance on pairwise similarity for pairs of entities, using LD and NC regularization.

Table 6.3 shows the results of performing this methodology, using both LD and NC regularizations for the JS corpus. While we were expecting a negative correlation between LCA distance and pairwise similarity, the regression tells us that there seems to be a signficant positive correlation. In Figure 6-10 we can visually observe that there is not actually much of a correlation in both cases. So from these few results, it is inconclusive if the latent entity hierarchy is doing something that is interpretable.



(a) Node Count Regularization



(b) Leaf Depth Regularization

Figure 6-10: Scatter plots of (LCA distance, pairwise simliarity) for all pairs of entities in JS corpus, under NC and LD regularization. There does not seem to be a strong correlation

6.6 Empirical Evaluation and Comparisions

Finally, we evaluate the performance of our method on the CDCR task. Unlike many other previous works which use a hand-tuned set of weights or a single feature [1, 22,

20], our method explicitly trains parameters from the corpus, so we will report 4-fold cross validation scores for our results. The comparision is shown in Table 6.4

	B-CUBED F_1
Bagga and Baldwin (1998) [1]	0.780
Rao et al. (2010) [20]	0.664
Singh et al. (2011)[22]	0.618
Latent Tree + SampleRank, LD regularization	**0.813**
Latent Tree + SampleRank, NC regularization	0.802
Latent Tree + CD, LD regularization	0.787

Table 6.4: Comparision of B-CUBED F_1 of previous work and our methods on the John Smith corpus.

We note that Bagga and Baldwin (1998) [1] is able to obtain a 0.846 F_1 measure by using complex article summary features, so the score we presented is their result using the full articles we do. Rao et al. (2010) [20] and Singh et al. (2011)[22] were more concerned with scalability of their CDCR method, so the low F_1 scores are likely a result of lack of proper parameter tuning.

For our methods, we observe that learning with SampleRank seems significantly better than Contrastive Divergence, which supports our observations in section 6.2 and makes sense because SampleRank has access to a domain-specific objection function. The difference between using LD and NC is small in this case, with LD doing performing slightly better.

These results show that our approach is competitive with previous works, outperforming them when we consider only the results using the same complexity of features we are using (recall, our feature were only bag-of-words, bag-of-names, and a useless entity-type feature). This probably the result of our method, first, learning the features in a systematic way and, second, capturing some structural information about the corpus in the form of structural regularization.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

The main goal of this thesis was to develop a new method for cross-document coreference, based on learning a latent tree structure over the mentions.

We augmented our model with a notion of regularizing complex tree structures. We observed that there are trade-offs between using different regularizations. For example, sum of leaf depth penalization seems to have lower variance than node count penalization. However, leaf depth regularization requires a size correction factor, which is only an approximate heuristic, while node count regularization needs no such thing to generalize.

To train the model parameters, we implemented a parameter estimation algorithm based on Contrastive Divergence and selectively deciding when to make parameter updates. To tune regularization constants, we implemented a form of ternary search. With these parameters, we used Metropolis Hastings inference to learn the latent tree structure over entities.

We then explored the behavior of our model on two datasets. We identified weaknesses in the model that become apparent during inference and showed how structural regularization helps overcome these problems. We analyzed additional effects structural regularization had on inference and attempted to quantify if the latent hierarchy over entities was interpretable.

Our method is competitive with other work in the literature, out-performing them if we consider using only a simple set of features. One explanation for this is that we don't only learn model parameters from the corpus, but also something inherent about the hierarchical structure of mentions, via the structural regularization constants.

7.2 Future Work

First, we were able to obtain very good results without using very sophisticated features. An obvious next step would be to include several other state-of-the-art features (such as LDA topic features) and seeing how our model performs. There are additionally many other structural statistics that can be explored (such as the number of children each node has) and characterized like we have done so in this thesis.

With the data becoming bigger and bigger everyday, many recent CDCR approaches have been developed to scale to corpuses with millions of mentions [22, 25, 20]. While our current approach does not scale to that magnitude, there is a clear path to make it scale. Because our main inference approach is based on Metropolis Hastings inference, which computes likelihood ratios using a very local part of the tree, we can follow the lead of [22, 25] and massively parallelize the inference procedure, because we can make proposals withining disjoint subtrees without affecting each other. Similarly, the parameter learning framework can be be parallelized by having several independent workers collecting gradient updates in parallel and sending them to a central master that combines them.

Finally, we can also make our approach an online algorithm for CDCR. Recall that our inference procedure starts with new mentions as children of the root of the tree. We thus can intermittently stream new mentions and add them as children of the root and continue performing inference. As long as the new mentions are generated similarly to the current mentions, we would not need to re-estimate model parameters. Furthermore, structural regularization constants can be size-corrected if necessary.

7.3 Summary

We have developed new perspective of coreference as the problem of learning a latent tree structure over the observed mentions. We show how to invoke Occam's razor to select simpler structures during inferene and provide methodologies for learning parameters for the model. By invoking on other fields, such structure learning, we see that it is possible to gain new insights for problems like cross-document coreference.

Appendix A

Derivations of Likelihood Ratio for Metropolis Hastings Proposals

Here we derive the form of the likelihood ratio for each of our Metropolis Hastings proposals.

A.1 Subtree Creation/Deletion

Recall the A (Subtree Create) move is as follows.

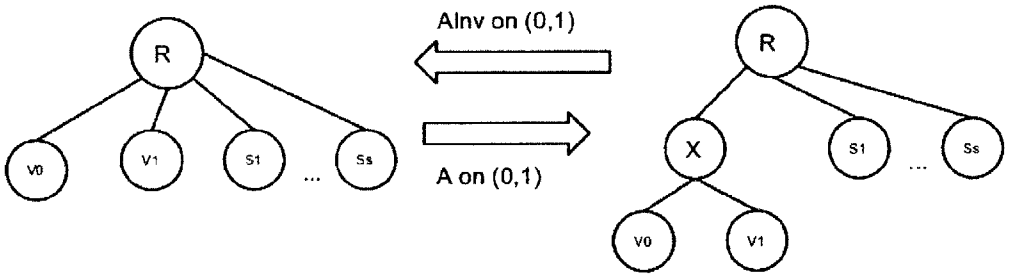


Figure A-1: Subtree Creation proposal and its inverse

Suppose we have a root vertex R with children V_0, V_1 and $S_1, \ldots, S_s$. Thus R has $1+1+s$ children. For ease of notation later, let us denote $S = \{S_1, \ldots S_s\}$, so $|S| = s$.

Furthermore, we have that

$$R = V_0 + V_1 + S_1 + \ldots S_s$$

where the sum is over data.

The structural differences (e.g. edges and nodes whose states have changed) before and after performing a Subtree Create action are summarized in Table A.1. We use the notation $R[A, B, C]$ to state that vertex R has children A, B, C and a $'$ indicates that the node is different. To get the structural differences for the Subtree Delete action, switch "Initial" and "Final". The node and edge potentials corresponding to nodes and edges in the structural differences are the only potentials that need to be re-computed for the likelihood ratio.

	Initial state	Final state
node	$R[v_0, v_1, S]$	$X[v_0, v_1], R'[X, S]$
edge	$(v_0, R), (v_1, R)$	$(v_0, X), (v_1, X), (X, R')$

Table A.1: Node and pairwise potentials that need to be re-computed for the Subtree Create/Subtree Delete actions.

A.2 Subtree Move Up/Down

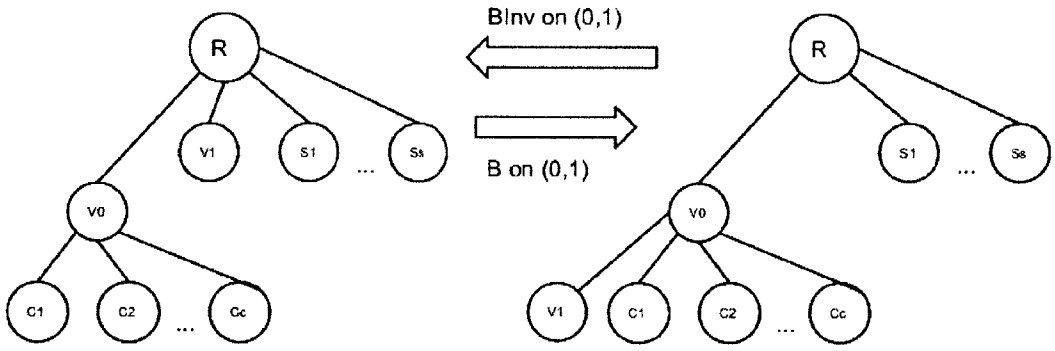


Figure A-2: Subtree Move Up/Down proposal action and its inverse

Again, we have a root node R with V_0, V_1 and $S_1, \ldots, S_s$. Additionally, we consider the node v_0 having children $C_1, \ldots C_c$. For ease of notation later, let us denote $C = \{C_1, \ldots C_c\}$. We thus additionally have

$$V_0 = C_1 + \ldots + C_c$$

where the sum, again, is over data.

In Table A.2 we show the corresponding node and edge structure differences before and after performing the Subtree Move Down action. Simlarily, to get the structure differences for the Subtree Move Up action, switch "Initial" and "Final" in the table.

node	Initial	$R[v_0, v_1, S], v_0[C]$
	Final	$R'[v_0, S], V_0'[v_1, C]$
edge	Initial	$(v_0, R), (v_1, R), \{(c_i, v_0) \forall c_i \in C\}$
	Final	$(v_0', R), (v_1, v_0'), \{(c_i, v_0') \forall c_i \in C\}$

Table A.2: Node and pairwise potentials that need to be re-computed for the Subtree Move Up/Down actions.

Appendix B

Online Algorithm to Update Pairwise F_1

Here we show how the Pairwise F_1 score can be updated efficiently, when the updates correspond to moving some subset of a cluster into another cluster. This algorithm uses $O(1)$ extra space for the counts of true positives, false positives, and false negatives. The full states of all the specific clusters do not need to be stored explicitly in one place, which is helpful for massively distributed setups.

We use the same notation as defined in Figure 2-1 and additionally define $count_{a \setminus b}(e)$ to be the number of mentions of entity e in the cluster $c_a \setminus c_b$.

1. Initialize TP, FP, FN to be the number pairs of true positives, false positives, and false negatives in the initial clustering (see Figure 2-1)

2. For any two clusters c_i, c_j and moving $c_s \subseteq c_i$ from c_i into c_j, update the above counts:

   (a) Compute the number of pairs of mentions in the original c_i that will go from positive (coreferent) to negative (not coreferent)

   $$\text{newFalsePairs} = \sum_{e \in E(c_i \cup c_j)} count_s(e) count_{i \setminus s}(e)$$

   (b) Compute the number of pairs of mentions in the original c_i that will go from negative to positive

   $$\text{newTruePairs} = \sum_{e \in E(c_i \cup c_j)} count_s(e) count_j(e)$$

   (c) Update the counts of true positives, false positives, false negatives as

   $$TP = TP + \text{newTruePairs} - \text{newFalsePairs}$$
   $$FN = FN + \text{newFalsePairs} - \text{newTruePairs}$$
   $$FP = FP + |c_s| \cdot (|c_j| - |c_i| + |c_s|) + \text{newFalsePairs} - \text{newTruePairs}$$

Figure B-1: Online Algorithm to Update Pairwise F_1 score

Bibliography

[1] A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the 17th international conference on Computational linguistics*, pages 79–85, 1998.

[2] S. Beheshti, S. Venugopal, S. Ryu, B Benatallah, and W. Wang. Big data and cross-document coreference resolution: Current state and future opportunities. Technical report, University of New South Wales, 2013.

[3] Seyed-Mehdi-Reza Beheshti, Srikumar Venugopal, Seung Hwan Ryu, Boualem Benatallah, and Wei Wang. Big data and cross-document coreference resolution: Current state and future opportunities. *CoRR*, abs/1311.3987, 2013.

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[5] Miguel A. Carreira-Perpinan and Geoffrey E. Hinton. On contrastive divergence learning. *Artificial Intelligence and Statistics*, 2005.

[6] Myung Jin Choi, Vincent Y. F. Tan, Animashree Anandkumar, and Alan S. Willsky. Learning latent tree graphical models. *Journal of Machine Learning Research*, 12:1771–1812, 2011.

[7] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theor.*, 14(3):462–467, September 2006.

[8] Aron Culotta, Michael Wick, Robert Hall, and Andrew Mccallum. First-order probabilistic models for coreference resolution. In *In Proceedings of HLT-NAACL 2007*, 2007.

[9] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[10] J. Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4(3):502–506, 1953.

[11] Nguy G. L. Machine learning approaches to coreference resolution, 2008.

[12] Peter Langfelder, Bin Zhang, and Steve Horvath. Defining clusters from a hierarchical cluster tree. *Bioinformatics*, 24(5):719–720, March 2008.

[13] Shuying Li, Dennis K. Pearl, and Hani Doss. Phylogenetic tree construction using Markov Chain Monte Carlo, 1999.

[14] Pavan Kumar Mallapragada, Rong Jin, and Anil K. Jain. Online visual vocabulary pruning using pairwise constraints. In *CVPR*, pages 3073–3080. IEEE, 2010.

[15] James Mayfield, David Alexander, Bonnie J. Dorr, Jason Eisner, Tamer Elsayed, Tim Finin, Clayton Fink, Marjorie Freedman, Nikesh Garera, Paul McNamee, Saif Mohammad, Douglas W. Oard, Christine D. Piatko, Asad B. Sayeed, Zareen Syed, Ralph M. Weischedel, Tan Xu, and David Yarowsky. Cross-document coreference resolution: A key technology for learning by reading. In *AAAI Spring Symposium: Learning by Reading and Learning to Read*, pages 65–70. AAAI, 2009.

[16] David Menestrina, Steven Euijong Whang, and Hector Garcia-Molina. Evaluating entity resolution results. *Proc. VLDB Endow.*, 3(1-2):208–219, September 2010.

[17] P. Murugavel. Improved hybrid clustering and distance-based technique for outlier removal. In *International Journal on Computer Science and Engineering (IJCSE)*, 2011.

[18] Chris Pal Pallika Kanani, Andrew McCallum. Improving author coreference by resource-bounded information gathering from the web. In *International Joint Conference on Artificial Intelligence*, 2007.

[19] Altaf Rahman and Vincent Ng. Supervised models for coreference resolution. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 968–977, 2009.

[20] Delip Rao, Paul McNamee, and Mark Dredze. Streaming cross document entity coreference resolution. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 1050–1058, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[21] Eric Shyu, Jay N. Baxter, John R. Frank, Max Kleiman-Weiner, and Dan A. Roberts. A tree-based approach to coreference. Technical report, Diffeo, Inc., 2013.

[22] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Proceedings of the 49th Annual Meeting of the Association for Computational*, pages 793–803, 2011.

[23] Sameer Singh, Michael L. Wick, and Andrew McCallum. Distantly labeling data for large scale cross-document coreference. *CoRR*, abs/1005.4298, 2010.

[24] M. Wick, K. Rohanimanesh, A. Culotta, and A. McCallum. Samplerank: Learning preferences from atomic gradients. In *Proceedings of NIPS Workshop on Advances in Ranking*, 2009.

[25] M. Wick Wick, S. Singh, and A. McCallum. A discriminative hierarchical model for fast coreference at large scale. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 379–388, 2012.

[26] René Witte and Sabine Bergler. Fuzzy Coreference Resolution for Summarization. In *Proceedings of 2003 International Symposium on Reference Resolution and Its Applications to Question Answering and Summarization (ARQAS)*, pages 43–50. Università Ca' Foscari, June 23–24 2003.