LEARNING AND INFERENCE IN WEIGHTED LOGIC WITH APPLICATION TO NATURAL LANGUAGE PROCESSING

A Dissertation Presented

by

ARON CULOTTA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2008

Computer Science

LEARNING AND INFERENCE IN WEIGHTED LOGIC WITH APPLICATION TO NATURAL LANGUAGE PROCESSING

A Dissertation Presented

by

ARON CULOTTA

Approved as to style and content by:

Andrew McCallum, Chair

Tom Dietterich, Member

David Jensen, Member

Jon Machta, Member

Robbie Moll, Member

Andrew Barto, Department Chair
Computer Science

For J.M. & L.M.

ACKNOWLEDGMENTS

Professionally, I am most indebted to my advisor. Andrew and I arrived at the University of Massachusetts in the same year, and I was very fortunate to have the opportunity to work with him. He has an undying, infectious enthusiasm for research and the rare ability to quickly distill a problem to its essence. His guidance is this single most important factor in my work.

I have also been fortunate to have a wonderful dissertation committee who have provided useful feedback along the way. Additionally, as Andrew's lab has rapidly grown in size, I have had many fruitful discussions and collaborations with other lab members, including Ron Bekkerman, Robert Hall, Pallika Kanani, Gideon Mann, Chris Pal, Charles Sutton, and Michael Wick. I am also thankful for my collaborations and discussions with researchers both inside and outside of UMass, including Jonathan Betz, Susan Dumais, Nanda Khambatla, David Kulp, Trausti Kristjansson, Ashish Sabharwal, Bart Selman, Jeffrey Sorenen, and Paul Viola.

Personally, I am grateful for having a wonderful, supportive family, without which I would never have been able to pursue my dreams. Thank you Mom, Dad, Stefan, and Alexis.

I would also like to acknowledge the various funding sources that supported me throughout my graduate career. This work was supported in part by the Center for Intelligent Information Retrieval, by a Microsoft Live Labs Fellowship, by the Central Intelligence Agency, the National Security Agency, the National Science Foundation under NSF grants #IIS-0326249 and #IIS-0427594, by the Defense Advanced Research Projects Agency, through the Department of the Interior, NBC, Acquisition Services Division, under contract

#NBCHD030010, and by U.S. Government contract #NBCH040171 through a subcontract with BBNT Solutions LLC. Any opinions, findings and conclusions or recommendations expressed in this material are my own and do not necessarily reflect those of the sponsor.

ABSTRACT

LEARNING AND INFERENCE IN WEIGHTED LOGIC WITH APPLICATION TO NATURAL LANGUAGE PROCESSING

MAY 2008

ARON CULOTTA

B.Sc., TULANE UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McCallum

Over the past two decades, statistical machine learning approaches to natural language processing have largely replaced earlier logic-based systems. These probabilistic methods have proven to be well-suited to the ambiguity inherent in human communication. However, the shift to statistical modeling has mostly abandoned the representational advantages of logic-based approaches. For example, many language processing problems can be more meaningfully expressed in first-order logic rather than propositional logic. Unfortunately, most machine learning algorithms have been developed for propositional knowledge representations.

In recent years, there have been a number of attempts to combine logical and probabilistic approaches to artificial intelligence. However, their impact on real-world applications has been limited because of serious scalability issues that arise when algorithms

designed for propositional representations are applied to first-order logic representations. In this thesis, we explore approximate learning and inference algorithms that are tailored for higher-order representations, and demonstrate that this synthesis of probability and logic can significantly improve the accuracy of several language processing systems.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

For many years, the field of artificial intelligence (AI) has been roughly divided into *symbolic* or *logical* approaches (logic programming, planning) and *statistical* approaches (graphical models, neural networks). Each approach has had important successes, but pursued in isolation their impact will be limited. Logic alone ignores the ambiguities of the real world, statistics alone ignores the relational complexities of the real world. This thesis explores issues in *weighted logic*, the synthesis of logic and probability.

Natural language processing (NLP) provides a promising application area for weighted logic. Because human language contains considerable ambiguity and relational complexity, it is likely that combining logical and statistical approaches will accelerate progress. However, NLP research has generally fluctuated from logical approaches (lambda calculus, rule-based parsers) to statistical approaches (Markov models, probabilistic classifiers). Weighted logic makes it possible to combine these two strains of research. As I will discuss in Chapter 3, there are many NLP problems that are currently partially solved using simple representations for reasons of efficiency. By moving to weighted logic, we can better model the complexities of language and therefore improve the prediction accuracy of NLP systems.

There is a small but growing line of research in AI aiming to unify probability and logic [41, 48, 83, 87, 76, 24, 40, 56, 73, 93]. Much of this foundational research has explored the semantics and expressivity of different formalisms for weighted logic. However, there have been few large-scale applications of probabilistic logic. This can be attributed primarily to

the scalability issues that flexible representation languages pose to traditional statistical inference algorithms. As a result, most existing applications do not completely utilize the expressive power that weighted logic provides.

While weighted logic provides an NLP researcher with a flexible language to represent language phenomena, its use within a real-world system poses significant computational challenges. The representation becomes a double-edge sword: from the user's perspective, it is quite easy to write down rules and dependencies in weighted logic; however, rules that are easy to write down may turn out to make learning and inference surprisingly computationally difficult. In fact, for most applications we are concerned with here, it is intractable to perform exact parameter estimation and inference in the resulting probabilistic model. The main goals of this thesis are (1) to develop effective approximations for learning and inference in weighted logic and (2) to show that with these approximations, weighted logic representations can improve the accuracy of NLP systems.

1.2 Contributions

Our contributions can be divided into three main components, outlined below.

1.2.1 Sampling for Weighted Logic

The first contribution is the application of ideas from Markov Chain Monte Carlo sampling to perform inference in weighted logic. We show how sampling methods are effective for weighted logic because of their scalable memory requirements. Most sampling algorithms operate over a small neighborhood of solution space, and therefore can avoid instantiating a potentially exponential number of random variables in the full model. In particular, we show how Metropolis-Hastings sampling is well-suited to inference in weighted logic because it relies on a *proposal distribution*, which can be understood as a transition function between solutions. By incorporating a bit of domain knowledge into this pro-

posal distribution, we can avoid instantiating a large number of deterministic constraints that characterize valid solutions.

1.2.2 Online Parameter Estimation for Weighted Logic

The second contribution is a general framework for parameter estimation in weighted logic. We can think of weighted logic as simply a set of assertions with weights associated with them, where the weight is correlated with the truth of the assertion. Parameter estimation is the problem of finding a reasonable assignment to these weights.

In this thesis, we propose a novel estimation algorithm, which we term *SampleRank*. SampleRank is an online, error-driven algorithm that was specially designed for use within sampling algorithms. Since we believe sampling algorithms are necessary for weighted logic, it follows that the learning algorithm should be designed with sampling in mind. The basic idea of SampleRank is to choose parameters that improve the accuracy of the sampler. The main advantages of SampleRank are:

- By using sampling as a subroutine, SampleRank avoids instantiating a potentially exponential number of variables.

- By updating parameters after each sample, SampleRank can more rapidly find good parameters assignments than batch algorithms (which require an entire pass through all training data before each update) or traditional online algorithms (which require an estimate of the best prediction before each update).

- Through the use of recently introduced large-margin optimization algorithms [19], SampleRank can avoid the large fluctuations in parameters often exhibited by online learning methods.

We perform a number of experiments on synthetic data as well as on real-world NLP problems to explore the behavior of SampleRank, and show that it can lead to state-of-the-art performance on a coreference resolution benchmark.

1.2.3 Multi-Task Inference

Finally, we present two separate algorithms for performing inference in weighted logic when the problem is a composition of related tasks. Many problems in artificial intelligence can be decomposed into loosely coupled subproblems (for example, pipelines of language processing tasks, or multi-agent systems). Our proposed algorithms take advantage of this structure by first solving subproblems independently, possibly with efficient, customized algorithms. Then, solution hypotheses are propagated between subtasks to converge upon a global solution. The first algorithm, *Sparse Max-Product*, performs message-passing between subtasks to reach an approximate global optimum. We demonstrate that this approach can be used to augment and in many cases outperform well-studied stochastic search algorithms such as MaxWalkSat [98]. The second algorithm, *Multi-Task Metropolis-Hastings*, is an extension of standard Metropolis-Hastings to multi-task problems. In particular, we use a specialized proposal distribution to rapidly search for highly-probable joint assignments for multiple tasks.

1.3 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 introduces terminology and notation for graphical models, and **Chapter 3** gives on overview of weighted logic, providing a number of examples from NLP to motivate this more flexible representation. **Chapter 4** proposes the uses of sampling algorithms for weighted logic and analyzes their suitability. **Chapter 5** introduces SampleRank and describes a number of possible instantiations of the algorithm obtained by using different parameter updates and sampling algorithms. The next three chapters provide experimental evidence for the effectiveness of SampleRank. **Chapter 6** provides a number of experiments on synthetic data to formalize some of the intuitions for why the algorithm works. **Chapter 7** then applies SampleRank to the problem of *named-entity recognition*, an important and widely-studied problem in NLP and information extraction. **Chapter 8** similarly

applies SampleRank to the problem of *coreference resolution*. In all of these examples, we find that combining the flexible representation of weighted logic with the SampleRank learning algorithm leads to higher prediction accuracy. In **Chapter 9** we present two algorithms for inference over multiple tasks, with some promising experimental results on real and synthetic data. Finally, in **Chapters 10** and **11**, we outline plans for future work and conclude. The discussion of related work is placed in subsections where it is relevant.

1.4 Previously published work

The ideas that led to SampleRank are present in our earlier published work, mainly motivated by the coreference resolution problem in both newswire documents and publication databases [20, 21, 23]. Additionally, in Culotta et al. [22], we introduced the idea of *Sparse Max-Product*, and presented experiments on satisfiability problems.

CHAPTER 2

BACKGROUND

This section introduces terminology and notation for graphical models. First, we will present a brief overview of graphical models, which have well-defined semantics for probabilistic reasoning over propositional representations. We will then review recent work that provides richer representational languages, such as first-order logic, to specify graphical models.

2.1 Graphical Models

Let $\mathbf{X} = \{X_1 \ldots X_n\}$ be a set of discrete random variables, and let $\mathbf{x} = \{x_1 \ldots x_n\}$ be an assignment to $\mathbf{X}$. Let $P(\mathbf{X})$ be the joint probability distribution over $\mathbf{X}$, and let $P(\mathbf{X} = \mathbf{x})$ be the probability of assignment $\mathbf{x}$, abbreviated $P(\mathbf{x})$, where $\sum_{\mathbf{x}} P(\mathbf{x}) = 1$.

While in general $P(\mathbf{X})$ may be an arbitrarily complex distribution, in practice conditional independence assumptions are made about the structure of $P(\mathbf{X})$ to enable efficient reasoning and estimation. These independence assumptions are realized by a *factorization* of $P(\mathbf{X})$ into a product of functions over subsets of $\mathbf{X}$. *Graphical models* provide a graph-theoretic approach to compactly representing a family of distributions that share equivalent factorizations.

Let $\mathbf{F} = \{f_1(\mathbf{x}^1) \ldots f_k(\mathbf{x}^k)\}$ be a set of functions called *factors*, where $\mathbf{x}^i \subset \mathbf{X}$ and $f : \mathbf{X}^n \mapsto \mathcal{R}^+$. A factor computes the compatibility of the assignment to its arguments. An *undirected graphical model* defines a family of probability distributions that can be expressed in the form

$$P(\mathbf{x}) = \frac{1}{Z} \prod_i f_i(\mathbf{x}^i) \tag{2.1}$$

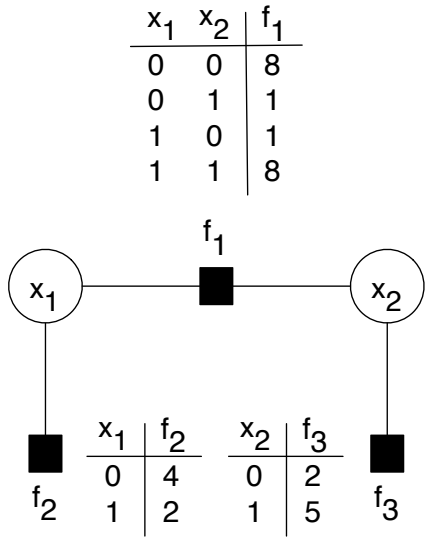

Figure 2.1. A factor graph with two binary variables $\{x_1, x_2\}$ and three factors $\{f_1, f_2, f_3\}$

where Z is a normalization constant (also known as a *partition function*) defined as

$$Z = \sum_{\mathbf{x}} \prod_{i} f_i(\mathbf{x}^i) \tag{2.2}$$

A *factor graph* [39] is a graph-based representation of the family of distributions defined by $\mathbf{X}$ and $\mathbf{F}$. A factor graph $G = (\mathbf{X}, \mathbf{F}, \mathbf{E})$ is a bipartite graph in which variable vertex $x_i \in \mathbf{X}$ is connected to factor vertex $f_j \in \mathbf{F}$ by edge $e_{ij} \in \mathbf{E}$ if and only if x_i is an argument to f_j (i.e., $x_i \in \mathbf{x}^j$). Figure 2.1 shows an example factor graph.

Different choices of $\mathbf{F}$ define different families of distributions. To define one specific distribution, we must fix the form of each factor f_i. In this work, we assume the form

$$f_i(\mathbf{x}^i) = \exp\left(\sum_j \lambda_j^i \phi_j^i(\mathbf{x}^i)\right) \tag{2.3}$$

where $\Lambda = \{\lambda_j^i\}$ is a vector of real-valued model parameters, and $\Phi = \{\phi_j^i\}$ is a collection of *feature functions* $\phi : \mathbf{X}^n \mapsto \mathcal{R}$. Substituting Equation 2.3 into Equation 2.1, we obtain

$$P(\mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i \lambda_i \phi_i(\mathbf{x})\right) \tag{2.4}$$

Note that to simplify notation we have dropped the factor index on $\lambda, \phi, \mathbf{x}$; it is implied that each ϕ_i only examines the variables that are arguments to its corresponding factor.

An undirected model with this choice of factors can be understood as the canonical form of the exponential distribution, with parameters Λ and sufficient statistics Φ. This model is commonly used in artificial intelligence and statistical physics, and is known as a *Markov random network* or *Markov random field* (MRF).

In many applications, we can divide $\mathbf{X}$ into variables that will always be observed (*evidence* variables $\mathbf{X}_e$) and variables that we would like to predict (*query* variables $\mathbf{X}_q$). In this case, it may be preferable to model the conditional distribution directly, rather than the joint distribution. This results in what is known as a *conditional random field* (CRF) [61, 107] :

$$P(\mathbf{x}_q|\mathbf{x}_e) = \frac{1}{Z(\mathbf{x}_e)} \exp\left(\sum_i \lambda_i \phi_i(\mathbf{x}_q, \mathbf{x}_e)\right) \tag{2.5}$$

Note that the normalizer $Z(\mathbf{x}_e)$ is now dependent on the assignment to the evidence variables:

$$Z(\mathbf{x}_e) = \sum_{\mathbf{x}_q} \exp\left(\sum_i \lambda_i \phi_i(\mathbf{x}_q, \mathbf{x}_e)\right) \tag{2.6}$$

The principal difference between MRFs and CRFs is that MRFs model the *joint* distribution over $\mathbf{X}_e$ and $\mathbf{X}_q$; whereas CRFs only model the *conditional* distribution of $\mathbf{X}_q$ *given* $\mathbf{X}_e$. CRFs often lead to more accurate predictions of $\mathbf{X}_q$ because MRFs "waste" modeling effort to predict observed variables. (For a more detailed discussion, see Sutton and McCallum [107].) In this thesis, we will focus on CRFs.

2.1.1 Inference

With the conditional distribution $P(\mathbf{x}_q|\mathbf{x}_e)$, we can answer a wide range of queries about $\mathbf{X}_q$. We refer to the procedure of answering queries as *inference*. A common type of

query alluded to in the previous section is one in which we observe the values $\mathbf{X}_e$ and wish to calculate the most probable assignment to unobserved variables $\mathbf{X}_q$, defined as

$$\mathbf{x}_q^* = \operatorname*{argmax}_{\mathbf{x}_q} p(\mathbf{x}_q|\mathbf{x}_e) \tag{2.7}$$

We will refer to this maximization problem as *most probable explanation* inference (MPE), or simply *prediction*.

A related type of query asks for the probability of a setting of unobserved variables, $P(\mathbf{X}_q = \mathbf{x}_q|\mathbf{x}_e)$. We refer to this as *probabilistic inference* (PI). A variant of PI instead calculates the marginal distribution over a single unobserved variable, $P(X_i = x_i, \mathbf{X}_q - X_i|\mathbf{x}_e)$. We refer to this as *marginal inference* (MI).

If the corresponding factor graph is acyclic (i.e., a tree), then MPE, PI, and MI can all be solved exactly using dynamic programming. Pearl [84] introduced the *sum-product* (or *belief propagation*) algorithm for PI, which can be understood as a protocol for passing "messages" between factor and variable vertices, where each message encodes information about the most likely assignments to subsets of variables. *Max-product* is a slight variation of sum-product that solves MPE. Sum-product and max-product are generalizations of the *forward-backward* and *Viterbi* algorithms developed for hidden Markov models [91].

If the factor graph contains cycles, exact inference can still be calculated by first constructing a *hypertree* to remove cycles, then performing message passing. This is often referred to as the *junction tree* algorithm [52] (or *clique tree* algorithm [62]). While exact, junction tree is impractical for many problems because the size of messages transmitted between two hyper-vertices grows exponentially in the number of variables they share. For this reason, approximate inference techniques are often used to solve large, real-world problems.

Two other families of approximations are *sampling* methods and *variational* methods. Sampling methods, such as Markov Chain Monte Carlo (MCMC), generate samples from a Markov chain that in the limit produces samples from the true distribution $P(\mathbf{X}_q|\mathbf{X}_e)$ [43].

Variational methods construct simpler factor graphs in which exact inference is tractable, then optimize a set of variational parameters to make this simpler distribution as close as possible to the original distribution [42]. A simple yet surprisingly effective variational approximation is *loopy belief propagation*. This iteratively applies standard sum-product on a cyclic graph, ignoring the "double-counting" of information that arises. Yedidia et al. [117] and Weiss and Freeman [114] have provided some theoretical justification for the success of this approximation. In practice, sampling methods often exhibit higher variance, while variational methods often exhibit higher bias. We will discuss these approximate inference methods in more detail in Chapter 4.

2.1.2 Learning

Learning in graphical models can refer either to *structure learning* or *parameter learning*. Structure learning optimizes the size and connectivity of the factor graph. Parameter learning optimizes the model parameters Λ. In this thesis we will focus on parameter learning.

Supervised learning assumes we are given a set of fully-observed samples of $\langle \mathbf{x}_q, \mathbf{x}_e \rangle$ pairs, called *training data*. From the training data, we can construct optimization objectives to guide learning. A common type of supervised learning is *maximum likelihood estimation* (MLE). Let $\mathcal{L}(\Lambda) = \log P_\Lambda(\hat{\mathbf{x}}_q|\hat{\mathbf{x}}_e)$ be the *conditional log-likelihood* of the training data given parameters Λ. Given a fixed factor graph G and training sample $\langle \hat{\mathbf{x}}_q, \hat{\mathbf{x}}_e \rangle$, MLE chooses Λ to maximize the conditional log-likelihood:

$$\Lambda^* = \operatorname*{argmax}_\Lambda \ \log P_\Lambda(\hat{\mathbf{x}}_q|\hat{\mathbf{x}}_e) = \operatorname*{argmax}_\Lambda \mathcal{L}(\Lambda) \tag{2.8}$$

Note that in an MRF, MLE would instead maximize the *joint* log-likelihood, $\log P_\Lambda(\hat{\mathbf{x}}_q, \hat{\mathbf{x}}_e)$.

When $\mathcal{L}(\Lambda)$ is convex, this optimization problem can be solved using hill-climbing methods such as gradient ascent or BFGS [63]. The gradient of the conditional log-likelihood is

$$\frac{\partial \mathcal{L}}{\partial \lambda_i} = \phi_i(\hat{\mathbf{x}}_q, \hat{\mathbf{x}}_e) - \sum_{\mathbf{x}'_q} \phi_i(\mathbf{x}'_q, \hat{\mathbf{x}}_e) P_\Lambda(\mathbf{x}'_q | \hat{\mathbf{x}}_e) \tag{2.9}$$

$$= \phi_i(\hat{\mathbf{x}}_q, \hat{\mathbf{x}}_e) - E_\Lambda[\phi_i(\mathbf{x}'_q, \hat{\mathbf{x}}_e)] \tag{2.10}$$

Maximizing the gradient has the appealing semantics of minimizing the difference between the empirical feature values and the expected feature values according to the model.

Note that the conditional log-likelihood must be computed many times during training until an optimum for Λ is reached. This can be computationally expensive because the expectation on the right-hand side requires summing over all possible assignments to $\mathbf{X}_q$ and performing probabilistic inference to compute $P_\Lambda(\mathbf{x}'_q | \hat{\mathbf{x}}_e)$. Thus, if inference is difficult, learning will be more so. There have been a number of approximate learning techniques proposed to mitigate this, including pseudo-likelihood [5], perceptron [17], and piecewise training [106]. We will discuss these approximate learning algorithms in more detail in Chapter 5.

2.2 Advanced Representations for Graphical Models

While graphical models provide a convenient formalism for specifying models and designing general learning and inference algorithms, in their simplest form they assume static, propositional data. That is, graphical models operate over variables, not *objects*. However, for many applications it is more natural to represent dependencies among objects, allowing us to specify properties of objects and relations between them.

In recent years, there have been a number of formalisms proposed to construct graphical models over these more advanced representations. *Relational Bayesian networks* [40] and *relational Markov networks* [110] are directed and undirected graphical models that can be specified using a relational database schema. *Relational dependency networks* [78] provide analogous relational semantics for dependency networks. Markov logic networks (MLNs) [93] extend this to RMNs to allow arbitrary first-order logic to be used as a template to construct an undirected graphical model.

Whereas the previous work can be understood as providing more complex representations for graphical models, a parallel line of research has investigated adding probabilistic information to first-order logic and logic programs [41, 48, 83, 87]. Gaifman [41] and Halpern [48] provide initial theoretical work in this area, and since then there have been a number of proposals for so called *first-order probabilistic languages*, including knowledge-based model construction [46], probabilistic extensions of inductive logic programming [76, 24, 56], and general purpose probabilistic programming languages [86, 73]. In Section 3.3, we will provide a more thorough description of these various weighted logic representations.

CHAPTER 3

WEIGHTED LOGIC

3.1 Overview

We use *weighted logic* as a general term to refer to any representational language that attaches statistical uncertainty to logical statements. These models have elsewhere been referred to as *First-Order Probabilistic Models* [66, 29]. We instead use the phrase weighted logic here to indicate that (a) the statistical uncertainty may not necessarily be modeled probabilistically, and (b) the representational language may not necessarily be first-order logic (e.g., second-order logic).

Note that while MRFs and CRFs can be interpreted as instances of weighted logic where the logical statements are expressed in propositional logic, in this thesis we will use the term weighted logic only to reference models using a representation more expressive than propositional logic, e.g., first-order logic.

The principal advantage of weighted logic representations over propositional representations is that by representing dependencies more abstractly, we can compactly specify dependencies over a large number of variables. For example, rather than specifying that "Bill Clinton lived in the White House" and "Ronald Reagan lived in the White House", we can specify that "All U.S. presidents lived in the White House". Of course, these assertions may not always hold, which is why we need to model their uncertainty.

As mentioned previously, there have been a large number of weighted logic formalisms proposed, each with different representational advantages and disadvantages. With few exceptions [87, 29, 73] the majority of weighted logic representations assume that learning and inference will be performed by first propositionalizing all assertions to create a graphi-

cal model, then using standard inference and learning techniques. In this section, we show how to compute such a propositionalization, and discuss the computational issues that arise. Specifically, we will show examples using Markov logic [93], although the issues arise in other formalisms as well.

A Markov logic network (MLN) can be understood as a template for constructing a factor graph. An MLN M consists of a set of $\langle R_i, \lambda_i \rangle$ pairs, where $R_i \in \mathbf{R}$ is a formula expressed in first-order logic and $\lambda_i \in \Lambda$ is its associated real-valued weight. The larger λ_i, the more likely it is that R_i will hold.

Given M and a set of observed constants, we can construct a factor graph $G = (\mathbf{X}, \mathbf{F}, \mathbf{E})$ that represents a distribution over *possible worlds*, that is, all possible truth assignments. We refer to the procedure of mapping an MLN to a factor graph as *grounding*, and it is accomplished as follows:

1. For each formula R_i, construct all possible *ground formula* $GF(R_i) = \{R_i^1 \ldots R_i^n\}$ by substituting in observed constants. For example, given a formula $R_1 : \forall x S(x) \Rightarrow T(x)$ and constants $\{a, b\}$, $GF(R_1) = \{S(a) \Rightarrow T(a), S(b) \Rightarrow T(b)\}$.

2. Convert each ground formula to a *ground clause*. For example, the ground formula $S(a) \Rightarrow T(a)$ is converted to $\neg S(a) \vee T(a)$. $S(a)$ and $T(a)$ are known as *positive ground literals*.

3. For each positive ground literal created by the previous step, create a binary variable vertex. We refer to the kth positive ground literal of ground clause R_i^j as $x_i^j(k)$. For example, if R_i^j is $\neg S(a) \vee T(a)$, then $x_i^j(0) \equiv S(a)$ and $x_i^j(1) \equiv T(a)$.

4. For each ground formula R_i^j, create a factor vertex f_i^j.

5. Place an edge between factor vertex f_i^j and each of its associated ground predicate variables $x_i^j(k)$. Let $\mathbf{x}_i^j$ refer to the set of variable vertices that are arguments to f_i^j (i.e., the set of ground predicates in the ground formula R_i^j).

6. Set the value of $f_i^j(\mathbf{x}_i^j) = \exp\left(\lambda_j \phi_i^j(\mathbf{x}_i^j)\right)$, where $\phi_i^j(\mathbf{x}_i^j) = 1$ if $\mathbf{x}_i^j$ satisfies clause R_i^j, and is 0 otherwise. For example, if $R_i^j \equiv \neg S(a) \vee T(a)$, then we can specify $f_i^j(\mathbf{x}_i^j)$ with the following table:

$S(a)$	$T(a)$	$f_i^j(S(a), T(a))$
0	0	e^{λ_i}
0	1	e^{λ_i}
1	0	1
1	1	e^{λ_i}

7. The final factor graph defines the Markov random field

$$P(\mathbf{x}) = \frac{1}{Z} \prod_i \prod_j f_i^j(\mathbf{x}_i^j) = \frac{1}{Z} \exp\left(\sum_i \sum_j \lambda_i^j \phi_i^j(\mathbf{x}_i^j) \right) \tag{3.1}$$

In the conditional setting, the corresponding conditional random field is

$$P(\mathbf{x}_q | \mathbf{x}_e) = \frac{1}{Z_{\mathbf{x}_e}} \exp\left(\sum_i \sum_j \lambda_i^j \phi_i^j((\mathbf{x}_i^j)_e, (\mathbf{x}_i^j)_q) \right) \tag{3.2}$$

The final factor graph therefore contains one binary variable vertex for each possible grounding of each predicate, and a factor vertex for each grounding of a first-order clause. If n is the number of observed objects and r is the number of distinct variables in the largest clause, then the final factor graph requires space $O(n^r)$.

This space complexity illustrates a critical problem in weighted logic: *while weighted logic provides flexible problem representation, it often results in graphical models that are too large to store in memory.* Since most learning and inference algorithms (exact or approximate) for graphical models assume the factor graph can at least be represented, this introduces new challenges for algorithm design.

3.2 Examples in Natural Language Processing

Given the computational challenges of weighted logic, a natural question is whether we really need such a complex representation for real-world applications. In this section, we present several natural language processing tasks that have previously been represented with propositional representations, and show examples of where a weighted logic representation would be beneficial. We will then show how the resulting factor graphs can quickly become intractable to store in memory. To present these examples, we use the notation $\mathbf{x}$ to refer to input variables (or observations) and $\mathbf{y}$ to refer to output variables (or predictions).

3.2.1 Named Entity Recognition

The input to *named-entity recognition* (NER) is a sentence $\mathbf{x}$ and the output is the entity label of each token, for example *Person, Organization*, etc. One of the most common and successful probabilistic approaches to this problem is to use a linear-chain conditional random field (CRF) [61]. This corresponds to the factor graph in Figure 3.1, without the factor connected by the dashed lines. The basic factor graph makes a first-order Markov assumption among output variables, therefore modeling the dependencies between pairs of labels, and each label and the input variables. These dependencies can be represented by simple clauses. The dependency among output variables is a clause $\mathit{NextLabel}(y_i, y_{i+1})$ which indicates the identity of adjacent labels. The dependencies over among the input variables can be represented by similar clauses $\mathit{WordLabel}(x_i, y_i)$. However, there are many interesting and potential useful dependencies that cannot be modeled by a linear-chain CRF. In the figure shown here, we have added an additional factor that indicates whether two tokens that are string identical have the same label or not (called a "skip-chain" dependency in Sutton and McCallum [104]). While the rule is quite simple to write down in first-order logic, it creates long cycles in the graph that greatly complicates inference. In Chapter 7, we will describe other such non-local features for NER and demonstrate that they can improve system accuracy when combined with SampleRank.

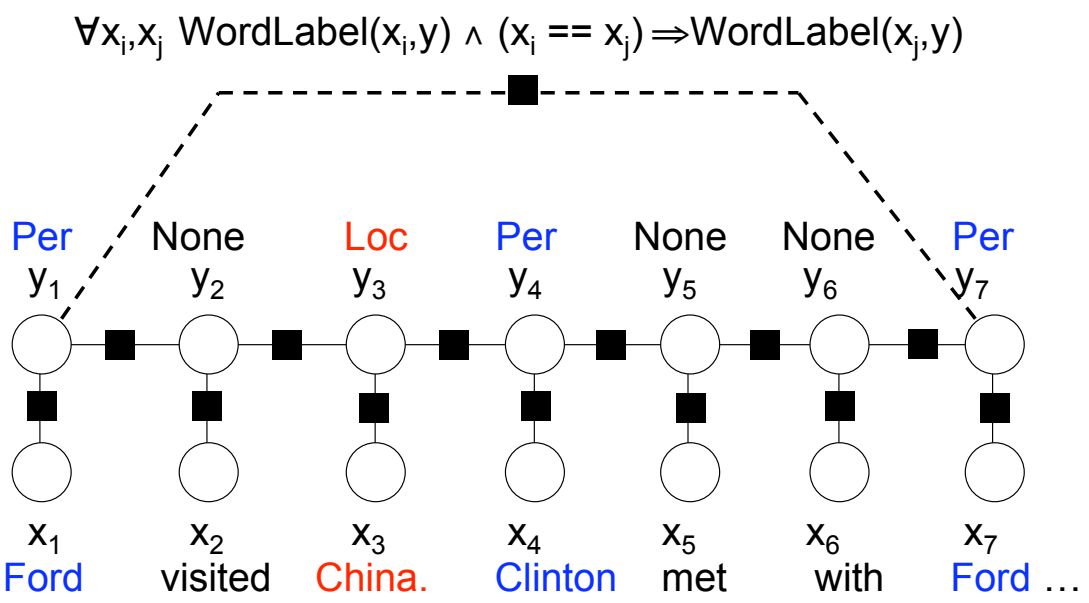

Figure 3.1. Example factor graph for the named-entity recognition problem. The standard probabilistic approach makes a first-order Markov assumption among the output variables to enable exact polynomial time inference. The additional dashed-line indicates a proposed dependency that makes dynamic programming infeasible. This so-called "skip-chain" dependency indicates whether all string-identical tokens in a document have the same label. This can be useful when there is less ambiguity in one context than in others.

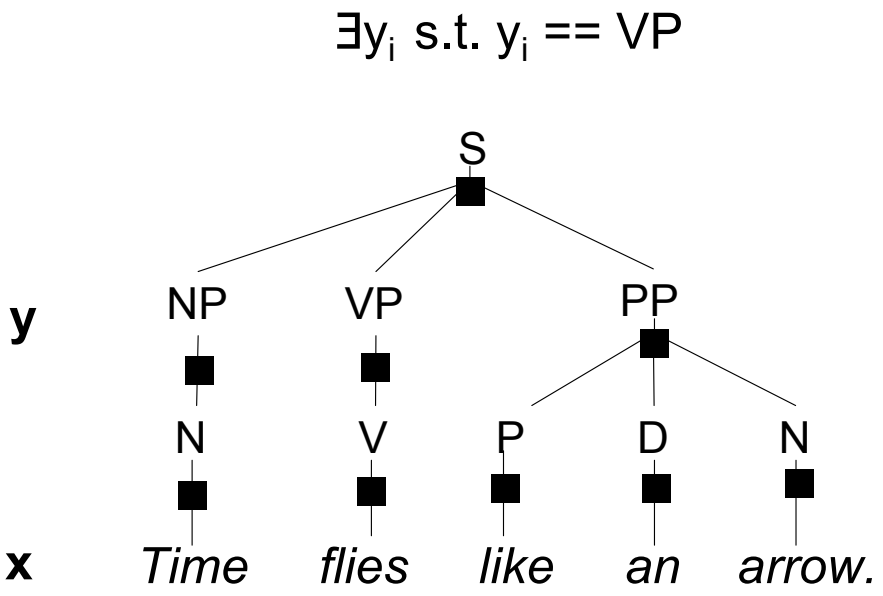

Figure 3.2. Example factor graph for the parsing problem. The input is a sentence and the output is a syntactic tree. The standard approach only models parent-children dependencies. A useful non-local dependency asks whether there exists some verb phrase in the predicted tree.

3.2.2 Parsing

The input to parsing is a sentence $\mathbf{x}$ and the output is a syntax tree $\mathbf{y}$ as shown in Figure 3.2. Parsing can be difficult to represent using standard factor graphs, since the structure of the graph is dynamic. Traditionally, probabilistic parsing is accomplished with probabilistic context-free grammars (PCFGs) [65], although there has been recent work using discriminative learning methods [109, 112]. In general, these models are restricted to local dependencies between a parent and its children to enable polynomial time dynamic programming algorithms for inference. However, it may be advantageous to include grandparent dependencies or dependencies that span the entire tree (e.g., does it contain more than one verb?). A notable attempt to incorporate these types of dependencies is found in Collins [16], who uses a post-processing classifier to rerank the output of a parser, thereby enabling the use of arbitrary features over the parse tree. However, this approach is limited by the quality of the initial parser.

3.2.3 Coreference Resolution

The input to *coreference resolution* is a set of mentions of entities (for example, the output of NER). The output is a clustering of these mentions into sets that refer to the same entity. This is an important step in information extraction, as well as for database management, where it goes under the name *record deduplication*.

A typical approach is to model dependencies between each pair of mentions. This can be described by rules in weighted logic such as $\textit{StringsMatch}(x_i, x_j) \Rightarrow \textit{Coreferent}(x_i, x_j)$. This rule states that if two mentions are string identical, then they are coreferent. However, there are many important dependencies that can only be captured when considering more than two mentions. As shown in Figure 3.3, it is important to know whether there exists a mention in a cluster that is not a pronoun. Since a personal pronoun should refer to some person mention, clusters that do not meet this criterion should be penalized.[1] In Chapter 8, we show that these higher-order dependencies can greatly improve the accuracy of coreference systems.

Modeling these dependencies requires a factor and y variable for each possible subset of $\mathbf{x}$. That is, we need a variable indicating whether every subset of $\mathbf{x}$ is coreferent. For almost any real-world dataset, we will be unable to store such a factor graph explicitly in memory.

Coreference resolution is an illustrative example for the computational demands of inference in weighted logic. In the next chapter, we will develop this example in more detail and show why sampling methods are required to avoid instantiating an exponential number of y variables.

---

[1] Note that this can be understood as a type of second-order logic, since we use existential predicates over subsets of objects.

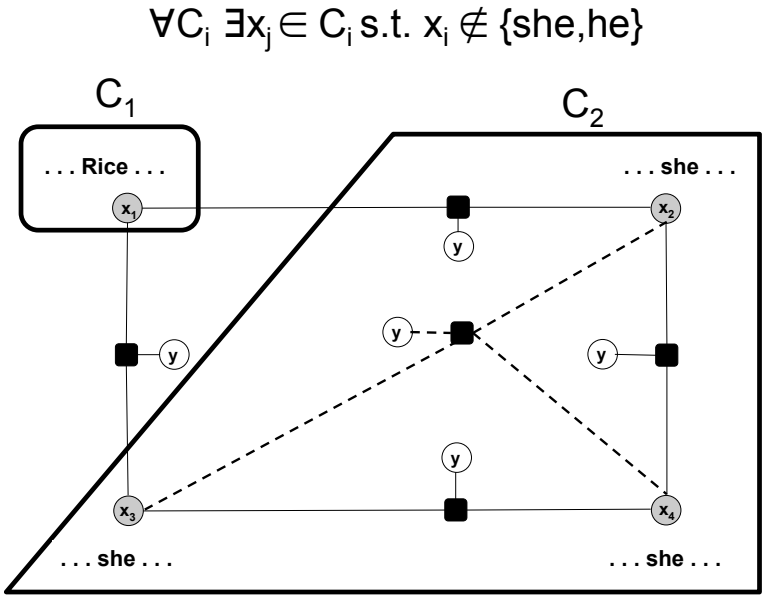

Figure 3.3. An example factor graph for coreference resolution. The input variables are mentions of entities or database records. The standard approach introduces binary output variables for each pair of mentions indicating whether they are coreferent. However, there are many interesting dependencies that require factors over *sets* of mentions, for example, whether or not the predicted cluster of mentions contains a non-pronoun mention. Such a dependency could be calculated by the factor connected with dashed lines.

3.3 Related Work

As mentioned in Chapter 2, there have been a large number of formalisms proposed to combine probability and logic, generally going by the name "First Order Probabilistic Models" (FOPL). For a thorough overview, see Milch [71]. Here, we give a brief survey of these formalisms.

One of the earliest contributions in this area is the work of Gaifman [41], which formalized many of the core concepts in what it means to define a probability distribution over a logical structure. While no formal language was proposed, this foundational work has led to many of the proposed representations develop over the past 20 years.

Halpern [48] proposes a probabilistic semantics for FOPL in which the output space is the set of all possible worlds, where a "world" is a grounding of all formulae. However, the model is parameterized only by a set of marginal constraints (e.g., $\forall x P(A(x)) = 0.4$), and so does not define a full joint distribution.

A number of FOPL proposals come out of the logic programming community. For example, Muggleton [76] present Stochastic Logic Programs, which define a distribution over the space of possible proofs from a logic program. Similarly, Kersting and Raedt [56] propose Bayesian logic programs, which defines a Bayesian network over logic programs, where variables correspond to statements that can be proved by the logic program.

Another strain of research can be understood as more complex ways of specifying graphical models. Paskin [83] presents Maximum Entropy Probabilistic Logic, which combines first-order logic statements within a maximum entropy model. Koller and Pfeffer [58] and Friedman et al. [40] present Probabilistic Relational Models, which are Bayes nets defined over frame structures. Similarly, Taskar et al. [110] introduce Relational Markov Networks (RMNs), which can be understood as a way of specifying an undirected graphical model using a database query language. Richardson and Domingos [93] propose Markov logic networks (MLNs), which augment RMNs by allowing a directed or undirected graphical model to be specified using first-order logic.

Finally, there have been a couple recent proposals of general purpose programming languages that contain probabilistic semantics. Pfeffer [86] presents a language called IBAL that allows probabilistic choices to be made during execution, thereby inducing a distribution over value assignments. Milch et al. [73] presents the BLOG language, which specifies distributions over possible worlds. The advantage of BLOG is that it enables the number of objects to be uncertain. Thus, one can reason about the values of objects that are not explicitly represented.

While each of these formalisms have contributed significantly to underlying semantic issues in weighted logic, the underlying computational issues of inference and learning still remain open problems. In the remainder of this thesis, we explore the challenges of learning and inference in weighted logic. We review related work in this area, propose new learning and inference algorithms, and evaluate them on NLP tasks.

CHAPTER 4

INFERENCE IN WEIGHTED LOGIC

4.1 Computational Challenges

There are two main computational challenges of performing inference in weighted logic. The first is the calculation of the normalization term $Z(\mathbf{x})$. Suppose for now that the entire set of output variables $\mathbf{y}$ can be stored in memory, and that the grounded factor graph represents the distribution $p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_i f_i(\mathbf{x}, \mathbf{y})$. Performing probabilistic inference requires calculating the normalization term $Z(\mathbf{x}) = \sum_y \prod_i f_i(\mathbf{x}, \mathbf{y})$. This is a well-studied problem in graphical modeling, and we will summarize the main approaches in Section 4.2.

The second computational challenge arises when the entire factor graph cannot be stored in memory. Recall the coreference example, in which the set of output variables $\mathbf{y}$ is exponential in the number of input variables $\mathbf{x}$. Even the coreference model that only considers pairs of mentions requires a quadratic number of output variables, which may be impractical for large datasets. In Section 4.3, we discuss how MCMC methods address this computational challenge because their computations are local in nature. In Section 4.4, we discuss how a particular type of MCMC algorithm, Metropolis-Hastings, is particularly memory-efficient in the presence of a large number of deterministic logical formulae, which appear in a number of NLP problems.

4.2 Markov-Chain Monte Carlo

When the graphical model corresponding to $p(\mathbf{y}|\mathbf{x})$ is a tree, $Z(\mathbf{x})$ can be computed exactly in polynomial time using the sum-product algorithm [84]. When the graph contains

cycles, the *junction tree* algorithm can be used to first map the graph to a hyper-tree, then use a variant of sum-product. However, the junction tree algorithm requires exponential space: If k is the size of the largest hyper-node in the hyper-tree (known as the *tree-width*), and D is the range of each variables, then the junction tree algorithm requires D^k space.

Unfortunately, as we have seen, the graphical models created by grounding a set of weighted logic formulae are rarely trees. We therefore will need to resort to approximate inference algorithms. There has been a voluminous amount of research on approximate inference algorithms for graphical models. These approximations can roughly be categorized as *variational* or *Monte Carlo*.

Variational algorithms (e.g., *mean field*, *loopy belief propagation*) approximate $p(\mathbf{y}|\mathbf{x})$ by a simpler distribution $q(\mathbf{y}|\mathbf{x})$, then use constrained optimization methods to find parameters of $q(\mathbf{y}|\mathbf{x})$ that make it a suitable substitute for $p(\mathbf{y}|\mathbf{x})$. (For a nice overview, see Ghahramani et al. [42].) While variational algorithms can be quite efficient, their performance can degrade in the presence of near-deterministic dependencies.

Monte Carlo algorithms (e.g., Metropolis-Hastings, Gibbs sampling, importance sampling) approximate the normalization term $Z(\mathbf{x})$ by drawing a set of samples $\{\mathbf{y}^{(1)} \ldots \mathbf{y}^{(T)}\} \sim p(\mathbf{y}|\mathbf{x})$. (For a nice overview, see Neal [77]). To calculate $Z(\mathbf{x})$, we can sum the unnormalized probabilities of these samples:

$$Z(\mathbf{x}) \approx \sum_t \prod_i f_i(\mathbf{y}^{(t)}, \mathbf{x}^{(t)})$$

Furthermore, to perform marginal inference for a single variable, i.e., $p(y_i|\mathbf{y}\ y_i, \mathbf{x})$, we can simply take the fraction of samples that satisfy the assignment of interest:

$$p(y_i = a|\mathbf{y}\ y_i, \mathbf{x}) \approx \frac{1}{T} \sum_{t=1}^{t=T} 1(y_i^{(t)} = a)$$

where $1(y_i^{(t)} = a)$ is an indicator function that is 1 if sample $\mathbf{y}^{(t)}$ assigns a to variable y_i, and is 0 otherwise.

A specific class of Monte Carlo algorithms is *Markov Chain Monte Carlo* (MCMC) [94]. The idea behind MCMC is to sample from $p(\mathbf{y}|\mathbf{x})$ by constructing a Markov chain whose stationary distribution is $p(\mathbf{y}|\mathbf{x})$. The states of the Markov chain correspond to assignments to the output variables $\mathbf{y}$. After a so-called "burn-in" period to allow the chain to approach its stationary distribution, states of the Markov chain are used as samples from $p(\mathbf{y}|\mathbf{x})$.

Below, we describe two popular MCMC algorithms: Gibbs sampling and Metropolis-Hastings.

4.2.1 Gibbs Sampling

Given a starting sample $\mathbf{y}^{(t)}$, Gibbs sampling generates a new sample $\mathbf{y}^{(t+1)}$ by changing the assignment to a single variable as follows:

- Choose a variable $y_i \in \mathbf{y}$ to be considered for a change (either chosen at random or according to a deterministic schedule).
- Construct the *Gibbs sampling distribution*:

$$p(y_i|\mathbf{y}^{(t)} \setminus y_i, \mathbf{x}) = \frac{\prod_j f_j(\mathbf{y}^{(t)} \setminus y_i, y_i, \mathbf{x})}{\sum_{y_i} \prod_j f_j(\mathbf{y}^{(t)} \setminus y_i, y_i, \mathbf{x})} \tag{4.1}$$

  This is just the distribution over the assignments to y_i assuming all other output variables take values in $\mathbf{y}^{(i)} \setminus y_i$.
- Sample an assignment $y_i \sim p(y_i|\mathbf{y}^{(t)} \setminus y_i, \mathbf{x})$.
- Set the new sample $\mathbf{y}^{(t+1)} \Leftarrow (\mathbf{y}^{(t)} \setminus y_i) \cup y_i$.

4.2.2 Metropolis-Hastings

In some applications, computing the Gibbs sampling distribution in Equation 4.1 is impractical. In these cases, we can instead use the *Metropolis-Hastings* algorithm [70, 50].

Algorithm 1 Metropolis-Hastings Sampler

Input:
target distribution $p(\mathbf{y}|\mathbf{x})$
proposal distribution $q(\mathbf{y}'|\mathbf{y}, \mathbf{x})$
initial assignment $\mathbf{y}^0$
for $t \leftarrow 0$ to NumberSamples **do**
  Sample $\mathbf{y}' \sim q(\mathbf{y}'|\mathbf{y}^t, \mathbf{x})$
  $\alpha = \min\{1, \frac{p(\mathbf{y}'|\mathbf{x})q(\mathbf{y}^t|\mathbf{y}', \mathbf{x})}{p(\mathbf{y}^t|\mathbf{x})q(\mathbf{y}'|\mathbf{y}^t, \mathbf{x})}\}$
  with probability α: $\mathbf{y}^{t+1} \Leftarrow \mathbf{y}'$
  with probability $(1 - \alpha)$: $\mathbf{y}^{t+1} \Leftarrow \mathbf{y}^t$
end for

Given an intractable *target distribution* $p(\mathbf{y}|\mathbf{x})$, Metropolis-Hastings generates a sample from $p(\mathbf{y}|\mathbf{x})$ by first sampling a new assignment $\mathbf{y}'$ from a simpler distribution $q(\mathbf{y}'|\mathbf{y}, \mathbf{x})$, called the *proposal distribution*. The new assignment $\mathbf{y}'$ is retained with probability

$$\alpha(\mathbf{y}'|\mathbf{y}, \mathbf{x}) = \min\left(1, \frac{p(\mathbf{y}'|\mathbf{x})q(\mathbf{y}|\mathbf{y}', \mathbf{x})}{p(\mathbf{y}|\mathbf{x})q(\mathbf{y}'|\mathbf{y}, \mathbf{x})}\right) \tag{4.2}$$

Otherwise, the original assignment $\mathbf{y}$ is kept. The Metropolis-Hastings algorithm is summarized in Algorithm 1.

Metropolis-Hastings can be understood as a rejection sampling method in which the target distribution is used to determine acceptance or rejection. The key mathematical trick to Metropolis-Hastings is that the acceptance probability $\alpha(\mathbf{y}'|\mathbf{y})$ requires only the *ratio* of probabilities from the target distribution $\frac{p(\mathbf{y}')}{p(\mathbf{y})}$. Because of this, the normalization terms cancel — we need only compute the unnormalized score for each assignment.

The proposal distribution $q(\mathbf{y}'|\mathbf{y})$ can be a nearly arbitrary distribution, as long as it can be sampled from efficiently and satisfies certain constraints to ensure that the stationary distribution of the Markov chain is indeed $p(\mathbf{y})$. (Typically, this is ensured by verifying that the *detailed balance* holds [77]). The proposal densities in Equation 4.2 are necessary to ensure detailed balance for asymmetric proposal distributions. Intuitively, these terms ensure that biases in the proposal distribution do not result in biased samples. We investigate this in more detail in Section 4.5.1.

4.3 Avoiding Complete Groundings using MCMC

Approximating the normalization term $Z(\mathbf{x})$ is a well-studied problem with many theoretical and empirical results. However, this is not the only difficulty of inference in weighted logic.

The second difficulty is that often the number of variables in the grounded factor graph is too large to store explicitly in memory. We therefore wish to perform inference in weighted logic without ever storing all possible output values $\mathbf{y}$ concurrently. We illustrate how MCMC can alleviate this problem using the following simplified example. Suppose we have the following weighted logic formulae:

$$A(x)[0.01]$$
$$A(x) \Rightarrow B(x)[0.2]$$
$$B(x) \Rightarrow C(x)[-0.9]$$
$$C(x) \Rightarrow D(x)[0.1]$$

Suppose that we are provided the constant a. Following the grounding algorithm outlined in Chapter 3, we would instantiate the factor graph shown in Figure 4.1.

The set of binary output variables is therefore $\{A(a), B(a), C(a), D(a)\}$, which we will abbreviate $\{y_1, y_2, y_3, y_4\}$. The intuition behind the memory efficiencies of MCMC sampling is as follows. Suppose the current assignment to the output variables is $\{0, 0, 0, 0\}$. If we use Gibbs sampling for inference, then we can reduce our computation to only consider *differences* between two assignments. For example, to sample a new assignment to $A(a)$, we need to consider two assignments: $\mathbf{y} = \{0, 0, 0, 0\}$ and $\mathbf{y}' = \{1, 0, 0, 0\}$. We can then calculate the Gibbs probability as follows:

$$p(y_1 = 1|\mathbf{x}, y_2 = y_3 = y_4 = 0) = \frac{\prod_i f_i(\mathbf{y} = \{1, 0, 0, 0\}, \mathbf{x})}{\prod_i f_i(\mathbf{y} = \{1, 0, 0, 0\}, \mathbf{x}) + \prod_i f_i(\mathbf{y} = \{0, 0, 0, 0\}, \mathbf{x})}$$

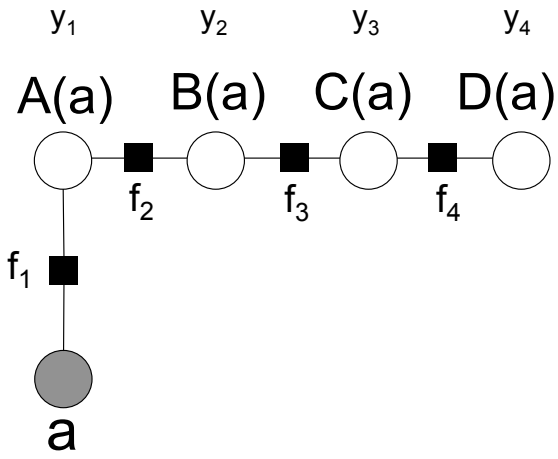

Figure 4.1. Factor graph created by grounding the four formulae in Section 4.3 for the constant a.

$$= \frac{f_1(y_1 = 1, x = a) * f_2(y_1 = 1, y_2 = 0)}{(f_1(y_1 = 1, x = a) * f_2(y_1 = 1, y_2 = 0)) + (f_1(y_1 = 0, x = a) * f_2(y_1 = 0, y_2 = 0))}$$

The savings here comes from the fact that to generate a sample for y_1, *we do not need to instantiate* y_3, y_4 since they have no effect on the sampling distribution.

Of course, in the worst case when all variables are set to true, we will still need to instantiate all variables. However, for many real-world problems, solutions are very sparse, i.e., most variable assignments are false. A similar observation is made in Singla and Domingos [99] in the context of designing memory-efficient satisfiability solvers for weighted logic. Whereas early work on Markov logic networks used Walksat (a weighted satisfiability solver) to perform prediction [93], that approach became impractical as the number of clauses grows large. Instead, Singla and Domingos [99] develop a *lazy* version of Walksat that only constructs clauses that are needed for the neighborhood of a single assignment.

4.4 Avoiding Grounding Deterministic Formulae using Metropolis-Hastings

Even with the local computations enabled by MCMC methods, it may be impractical to explicitly instantiate all the factors needed to compute the sampling distribution. This

is particularly problematic when there exist a number of deterministic rules to ensure the feasibility of a solution.

For example, consider again the coreference resolution problem described in Chapter 3. An important formula we omitted from our initial discussion is one enforcing transitivity among output variables. That is, if a is coreferent with b, and b is coreferent with c, then a should be coreferent with c. This can be expressed in first-order logic as:

$$\forall(x, y, z) \text{ Coreferent}(x, y) \wedge \text{Coreferent}(y, z) \Rightarrow \text{Coreferent}(x, y, z)$$

Because this formula must be satisfied for a coreference solution to be valid, a weight of negative infinity should be associated with it.

This transitivity formula increases the size of our factor graph by a polynomial factor. If $\mathbf{x}$ is the set of input mentions, and $n = |\mathbf{x}|$, then number of binary coreference variables is $O(n^2)$ and the number of transitivity factors is $O(n^3)$.

Even this relatively low-order polynomial complexity can be problematic. For example, coreference resolution is often used to deduplicate records in databases, where n can easily be in the order of thousands or millions.

In many common approaches to inference in weighted logic, such deterministic factors are explicitly instantiated. For example, Singla and Domingos [99] construct a weighted satisfiability problem from the model in Figure 4.2 containing transitivity clauses with very large weights to encourage valid solutions. There are two issues with their approach: (1) as discussed, the number of transitive clauses can become impractical to store, and (2) since the transitivity weights are not set to positive infinity, there is no guarantee that the final solution will actually be a valid solution.

We can address both of these issues using Metropolis-Hastings. The observation is quite simple. As long as the following two conditions are satisfied, we can guarantee we will only sample valid solutions:

- The initial sample is a valid solution.

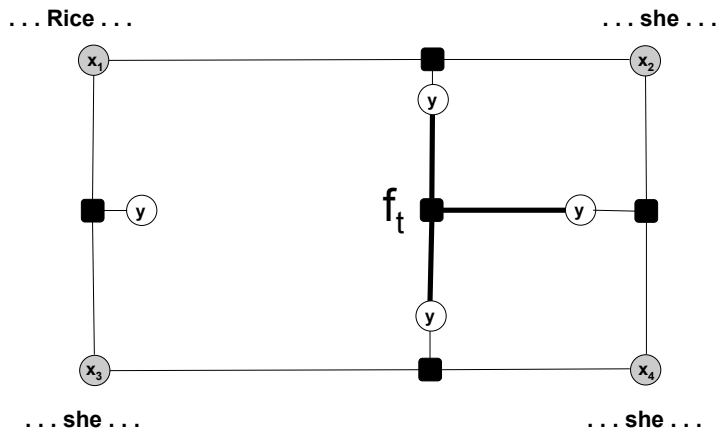

Figure 4.2. A factor graph for the coreference problem. In addition to factors for each pair of mentions, deterministic factors f_t must be introduced for triples of y variables to ensure that transitivity is enforced.

- If $\mathbf{y}$ is a valid solution, then the proposal distribution $q(\mathbf{y}'|\mathbf{y}, \mathbf{x})$ will always return a valid solution.

Because the proposal distribution is domain dependent, it is relatively straight-forward to ensure that these conditions are satisfied. For example, for coreference resolution, we can design a proposal distribution that only generates new samples by merging or splitting of clusters predicted by the previous sample. This ensures each sample is a valid clustering (i.e., satisfies transitivity).

In this manner, we can design inference algorithms that are guaranteed to return valid solutions without requiring explicit representation of a polynomial number of deterministic factors.

4.5 From Sampling to Prediction

MCMC methods are designed to solve probabilistic inference, i.e., computing the value $p(\mathbf{y}|\mathbf{x})$. For many real-world problems, however, we are also interested in prediction, i.e., finding the most probable assignment to $\mathbf{y}$.

There are three common methods of converting a sampling algorithm for probabilistic inference into a prediction algorithm, as discussed in Robert and Casella [94].

- **Select the best sample.** This straight-forward method simply performs sampling as usual and keeps track of the highest scoring sample seen. This sample is then returned as the most probable solution.

- **Fixed temperature.** This approach introduces a *temperature* parameter (fixed in advance) to control the tradeoff between maximization and exploration. For Gibbs sampling over binary variables, the sampling distribution changes as follows. To generate a new assignment for y_i from current state $\mathbf{y}^{(t)}$, we first define

$$\Delta(y_i = 1) = \prod_j f_j(\mathbf{y}^{(t)} \setminus y_i, y_i = 1, \mathbf{x}) - \prod_j f_j(\mathbf{y}^{(t)} \setminus y_i, y_i = 0, \mathbf{x})$$

and similarly for $\Delta(y_i = 0)$. This value is the change in model score introduced by the new sample.

Let $U(0, 1)$ be a number sampled uniformly at random in the range $(0, 1)$. Then a new assignment $y_i = 1$ is accepted at temperature T if the following condition is satisfied:

$$\exp\left(\frac{\Delta(y_i = 1)}{T}\right) > U(0, 1)$$

Similarly, for the Metropolis-Hastings algorithm, a new sample $\mathbf{y}'$ is accepted if the following is satisfied:

$$\exp\left(\frac{p(\mathbf{y}'|\mathbf{x})q(\mathbf{y}|\mathbf{y}', \mathbf{x}) - p(\mathbf{y}|\mathbf{x})q(\mathbf{y}'|\mathbf{y}, \mathbf{x})}{T}\right) > U(0, 1)$$

Note that in both cases, the samplers will always accept assignments that improve the scoring function. Assignments that decrease the scoring function are accepted with some probability. As the temperature approaches 0, this probability shrinks, and the

sampling algorithm becomes more similar to a greedy search algorithm. At $T = 0$, only greedy moves are accepted.

- **Variable temperature.** The final approach is a variation on the fixed temperature algorithms. Rather than specifying a fixed temperature ahead of time, the temperature is gradually decreased over time according to some (typically geometric) schedule, encouraging more exploration during the beginning of sampling, and more maximization toward the end. This is the idea behind the well-studied *Simulated Annealing* method [57]. While theoretical analysis of Simulated Annealing has been elusive, it has shown to be valuable for a number of optimization tasks.

Note that for each of the above methods, we are no longer sampling from the underlying distribution $p(\mathbf{y}|\mathbf{x})$. Instead, we are using knowledge of this underlying distribution to guide search for the most probable assignment.

4.5.1 Metropolis-Hastings for Prediction

In the experiments presented in the following sections, we make a greedy approximation for prediction (i.e., sampling with fixed temperature set to 0). In preliminary experiments, we did not observe significantly different results using less greedy methods. Given this choice, it is worth examining whether we need any of the MCMC machinery to perform prediction. For example, for Metropolis-Hastings, it is tempting to ignore the value of the proposal distribution $q(\mathbf{y}'|\mathbf{y}, \mathbf{x})$ in the acceptance calculation, instead simply performing greedy search on the target distribution $p(\mathbf{y}|\mathbf{x})$. The proposal distribution is included in the acceptance calculation mainly to ensure that the Markov chain converges to the correct equilibrium for asymmetric proposal distributions. While this is necessary for sampling, it can also have an effect on prediction. In particular, the ratio of proposal densities can prevent the chain from searching a biased subset of the output space.

In this section, we present results on a synthetic dataset to explore this issue. We generate clustering problems and compare prediction accuracy between standard Metropolis-

Hastings and a version of Metropolis-Hastings that ignores the proposal distribution density.

For each trial, we sample C clusters from a Dirichlet ($\alpha = 2$). The number of clusters C is sampled from a Gaussian with mean 4 and variance 2. Each cluster is represented by a multinomial distribution sampled from the Dirichlet. We sample a total of 50 multinomial data points, each with dimension $2C$.

We construct a *split-merge* proposal distribution $q(\mathbf{y}'|\mathbf{y}, \mathbf{x})$ that randomly merges and splits existing clusters. We sample from $q(\mathbf{y}'|\mathbf{y}, \mathbf{x})$ as follows:

- Let q_s be the prior probability that a splitting operation is sampled.
- Let $q_m = 1 - q_s$ be the prior probability that a merging operation is sampled.
- Sample the *proposal type* $\in \{$*merge*, *split*$\}$ according to q_s, q_m.
- If the *proposal type* is *merge*:
  - Select two clusters u.a.r. from $\mathbf{y}$ and merge them to create $\mathbf{y}'$.
  - Let c be the number of clusters in $\mathbf{y}$.
  - Let d be the number of clusters in $\mathbf{y}'$ with size > 1.
  - The forward probability is $q(\mathbf{y}'|\mathbf{y}) = q_m * \frac{1}{c(c-1)}$
  - The backward probability is $q(\mathbf{y}|\mathbf{y}') = q_s * \frac{1}{d}$
- If the *proposal type* is *split*:
  - Select a cluster u.a.r.
  - Partition the cluster into two clusters u.a.r. to generate $\mathbf{y}'$.
  - Let c be the number of clusters in $\mathbf{y}'$.
  - Let d be the number of clusters in $\mathbf{y}$ with size > 1.
  - The forward probability is $q(\mathbf{y}'|\mathbf{y}) = q_s * \frac{1}{d}$

– The backward probability is $q(\mathbf{y}|\mathbf{y}') = q_m * \frac{1}{c(c-1)}$

Thus, the proposal distribution generates random splits and merges of the clusters, and the proposal density is a product of the split (or merge) prior probabilities and a uniform density.

Rather than learn the parameters of the target distribution, we use the value of $p(\mathbf{y}|\mathbf{x})$ so we can isolate the effects of different prediction algorithms. We compare two prediction algorithms:

- **MH:** Metropolis-Hastings with the standard acceptance calculation. To perform prediction, we generate 100 samples and select the highest scoring assignment.
- **M:** Metropolis-Hastings where the terms $q(\mathbf{y}|\mathbf{y}', \mathbf{x})$ and $q(\mathbf{y}'|\mathbf{y}, \mathbf{x})$ are omitted from the acceptance calculation. Prediction is performed the same as in **MH**.

To evaluate the effects of different forms of the proposal distribution, we vary the prior probability of a split move $q_s \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

Figure 4.3 displays the BCubed F1 score [2] for the best solution found from each method. Each data point is an average of 10 random samples.

We note two conclusions from this figure. First, the quality of the proposal distribution can have a dramatic effect on the quality of the prediction. Because the synthetic data contained clusters of approximately 10 nodes, an proposal distribution that is biased towards oversplitting will generate lower quality solutions.

Second, the proposal distribution density appears to help the sampler locate high-probability solutions efficiently. Upon closer inspection, we find that **M** often has a very high acceptance rate, even though the proposal distribution generates changes uniformly. By multiplying the ratio of proposal densities, **MH** rejects a number of low-quality samples.

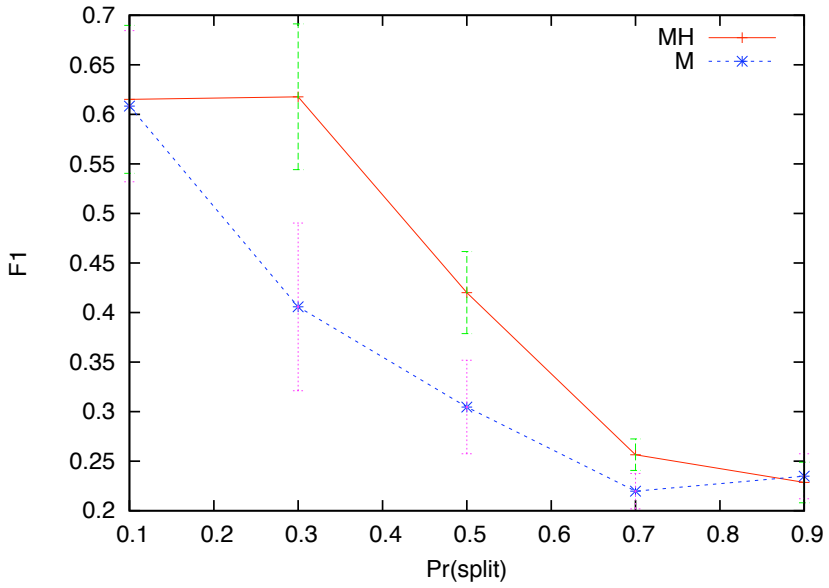

Figure 4.3. Accuracy of prediction for Metropolis-Hastings using the proposal density (**MH**) versus not using the proposal density (**M**). **MH** appears to be more robust to biases in the proposal distribution.

4.6 Related Work

Milch et al. [73] also uses Metropolis-Hastings for use within a first-order probabilistic model. The main difference here is that we are using conditional models (rather than joint models).

Viewed as a way to avoid grounding an entire network from a first-order representation, our approach is similar in spirit to *lifted inference* [29], an inference algorithm for weighted logic that reasons about the network without grounding. While lifted inference does no grounding, here we ground only the amount needed to distinguish between pairs of assignments. Lifted inference is a promising direction; however, in many domains it is necessary to make inferences about *specific* input nodes, rather than over the entire population. It is not clear how current versions of lifted inference can address this, although approximate methods may be possible.

Singla and Domingos [99] present a "lazy" prediction algorithm called LazySAT that avoids grounding the entire network. Their algorithm operates by only instantiating clauses

that may become unsatisfied at the next iteration of prediction. Our work can be understood as a lazy algorithm for parameter estimation, but is not tied to one specific prediction algorithm. However, LazySAT does not must still explicitly ground deterministic factors, while this can be avoided with Metropolis-Hastings sampling.

Poon and Domingos [88] present the MC-SAT algorithm, a version of Gibbs sampling that uses a satisfiability solver as a subroutine. MC-SAT has been shown to perform quite well in the presence of near-deterministic factors. However, this approach suffers from the problems discussed in Section 4.4; namely, all deterministic factors must be explicitly represented. A recent extension, Lazy MC-SAT [90], combines MC-SAT with LazySat. While this reduces the number of ground variables, the deterministic factors must still be stored as in LazySAT.

CHAPTER 5

PARAMETER LEARNING IN WEIGHTED LOGIC

Parameter learning (or simply *learning*) is the task of setting the real-valued weights Λ that parameterize each factor. In this thesis, we assume the *supervised learning* setting, in which we are provided with n *training examples* $D = \{(\mathbf{y}^{(1)}, \mathbf{x}^{(1)}) \dots (\mathbf{y}^{(n)}, \mathbf{x}^{(n)})\}$.

As described in Chapter 2, inference (either probabilistic inference or prediction) is typically called as a subroutine to learning algorithms. Therefore, the approximations discussed in the previous chapter will need to be used during learning as well.

In this chapter, we will first describe several existing approximate learning algorithms that can be applied to learning in weighted logic. We then discuss the drawbacks of these previous approaches, and propose a new learning framework called *SampleRank* that attempts to overcome some of these drawbacks.

5.1 Approximate Learning Algorithms

As discussed in Chapter 2, exact maximum likelihood learning is intractable for most real-world problems for which we would like to use weighted logic. Below we briefly outline three basic approaches to approximate learning.

5.1.1 Learning with Approximate Expectations

Recall the gradient of the conditional likelihood objective presented in Chapter 2, which can be interpreted as the difference between empirical feature counts and expected feature counts according to the model with current parameters Λ:

$$\frac{\partial \mathcal{L}}{\partial \lambda_i} = \phi_i(\hat{\mathbf{y}}, \hat{\mathbf{x}}) - \sum_{\mathbf{y}'} \phi_i(\mathbf{y}', \hat{\mathbf{x}}) P_\Lambda(\mathbf{y}'|\hat{\mathbf{x}}) \tag{5.1}$$

where $\hat{\mathbf{x}}$ are the observed variables for a training example, $\hat{\mathbf{y}}$ is the true assignment to the output variables, and ϕ are feature functions.

When the sum over assignment to $\mathbf{y}$ in the second term cannot be calculated exactly, a simple approximation is to replace the summation over all assignment with a summation over a sample of assignments. These samples can be generated using the MCMC algorithms of Section 4.2.

Unfortunately, running MCMC until convergence after each parameter update can quickly become impractical. Recently, Hinton [51] proposed *contrastive divergence*, a variant of the previous learning algorithm that starts from the true assignment $\hat{\mathbf{y}}$ and generates only a few samples. A KL-divergence calculation is then used to generate the gradient direction. While contrastive divergence has been shown to be quite useful in some domains, in this thesis we will focus on online learning methods because of their simplicity and efficiency.

5.1.2 Online Learning

An alternative approach to approximating the expected counts with MCMC samples is to instead approximate them with the best assignment found with the current parameters. This is the approach of *online learning*, in which the parameters are updated after each prediction is generated. This often results in simple and efficient updates, which makes online learning well-suited to the large models that are common in weighted logic.

5.1.2.1 Perceptron

The perceptron update proposed by Rosenblatt [95] is one of the earliest known learning algorithms. The original algorithm was extended to the *averaged* and *voted perceptron* by Freund and Schapire [38] to improve stability of the algorithm, and was extended to structured classification problems by Collins [17].

Algorithm 2 Averaged Perceptron

Input:
Initial parameters Λ
Distribution $p(\mathbf{y}|\mathbf{x})$
Training examples $D = \{(\mathbf{y}^{(1)}, \mathbf{x}^{(1)}) \dots (\mathbf{y}^{(n)}, \mathbf{x}^{(n)})\}$
Sum of parameters γ initialized to the vector $\bar{0}$
for $k \leftarrow 0$ to NumberIterations N **do**
  for $j \leftarrow 0$ to n **do**
    Estimate $\mathbf{y}' \approx \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}^{(j)})$
    if $\mathbf{y}' \neq \mathbf{y}^{(j)}$ **then**
      Update $\Lambda^{t+1} \Leftarrow \Lambda^t + \left(\Phi(\mathbf{y}^{(j)}, \mathbf{x}^{(j)}) - \Phi(\mathbf{y}', \mathbf{x}^{(j)})\right)$
      Add to sum: $\gamma \Leftarrow \gamma + \Lambda^{t+1}$
    end if
  end for
end for
return $\gamma/(Nn)$

Let $\Phi(\mathbf{y}, \mathbf{x})$ be the vector of features $\{\phi_1(\mathbf{y}, \mathbf{x}) \dots \phi_n(\mathbf{y}, \mathbf{x})\}$ defined by the factor functions of the model. The perceptron algorithm uses a simple additive update to adjust the parameters when a mistake is made on a training instance. Algorithm 2 gives the details of the algorithm. The basic idea is that whenever a mistake is made by the prediction algorithm, the parameter vector is reduced by the features that occur only in the mistaken assignment and incremented by the features that occur only in the true assignment. The average of these updates is returned to reduce the variance of the resulting model. Collins [17] shows that if the training examples are linearly separable, then averaged perceptron will converge to perfect accuracy on the training data.

5.1.2.2 MIRA

The Margin-Infused Relaxation Algorithm (MIRA) has been proposed recently by Crammer et al. [19]. MIRA attempts to address two issues with the perceptron update. First, rather than simply performing an additive update, a hard constraint is used to ensure that after the update the true assignment has a higher model score than the incorrect assignment by some margin M. Second, to reduce parameter fluctuations, the vector norm of the difference between successive updates is minimized. The resulting algorithm is the same

as the perceptron algorithm presented in Algorithm 2, except that line 6 is changed to the following:

$$\Lambda^{t+1} = \underset{\Lambda}{\operatorname{argmin}} ||\Lambda^t - \Lambda||^2 \;\; s.t. \;\; \prod_i f_i(\mathbf{y}^{(j)}, \mathbf{x}^{(j)}) - \prod_i f_i(\mathbf{y}', \mathbf{x}^{(j)}) \geq M \tag{5.2}$$

MIRA with a single constraint can be efficiently solved in one iteration of the Hildreth and D'esopo method [14]. Note that the use of only the top scoring predicted assignment is an approximation to the exact large-margin update, which would require a constraint for every other possible assignment. This approximation has been shown to perform well for dependency parsing in McDonald and Pereira [69].

To further reduce effects of parameter fluctuations, we again average the parameters created at each iteration as in the voted perceptron algorithm.

5.1.3 Reranking

Finally, we mention a relatively simple learning algorithm that has nonetheless exhibited quite competitive performance on a number of NLP problems. The general idea of *reranking* is the following: Given some complex distribution $p(\mathbf{y}|\mathbf{x})$, construct a simpler model $q(\mathbf{y}|\mathbf{x})$. We assume that for any example $\mathbf{x}^{(i)}$, we can compute the top K assignments according to $q(\mathbf{y}|\mathbf{x}^{(i)})$, which we will refer to as $K(q, \mathbf{x}^{(i)})$. The learning phase proceeds by training a classifier to rank the true assignment $\mathbf{y}^{(i)}$ higher than the competing K-best predictions $K(q, \mathbf{x}^{(i)})$. (If $\mathbf{y}^{(i)} \in K(q, \mathbf{x}^{(i)})$, it is removed from the set during training.) The features of this reranking classifier include all those in the factors of the original, complex distribution $p(\mathbf{y}|\mathbf{x})$ that made the model intractable. At testing time, the top assignments are computed for each testing example according to q, and these assignments are then reranked by the induced classifier. Algorithm 3 shows pseudo-code for this approach.

Although any classifier could be used for reranking, in this thesis we use a maximum-entropy (i.e., logistic regression) classifier. Traditional multi-class logistic regression defines the conditional distribution

Algorithm 3 Reranking Estimation

1: **Input:**
Tractable distribution $q(\mathbf{y}|\mathbf{x})$ with initial parameters Θ
Reranking classifier C with parameters Λ
Training examples $D_1 = \{(\mathbf{y}^{(1)}, \mathbf{x}^{(1)}) \ldots (\mathbf{y}^{(n)}, \mathbf{x}^{(n)})\}$
Development examples $D_2 = \{(\mathbf{y}^{(1)}, \mathbf{x}^{(1)}) \ldots (\mathbf{y}^{(m)}, \mathbf{x}^{(m)})\}$
2: Estimate the parameters Θ of $q(\mathbf{y}|\mathbf{x})$ on D_1 using some possibly exact learning algorithm.
3: Initialize the set of reranking training examples $R \Leftarrow \emptyset$.
4: **for** $(y^{(j)}, \mathbf{x}^{(j)}) \in D_2$ **do**
5: Generate the top assignments $K(q, \mathbf{x}^{(j)})$ from $q(\mathbf{y}|\mathbf{x}^{(j)})$
6: Add training example: $R \Leftarrow R \cup \left(K(q, \mathbf{x}^{(j)}), \mathbf{y}^{(i)}\right)$
7: **end for**
8: **return** $q(\mathbf{y}|\mathbf{c})$ and C

$$p(y|\mathbf{x}) = \frac{\exp \sum_i \lambda_i \phi_i(y, \mathbf{x})}{\sum_{y'} \exp \sum_i \lambda_i \phi_i(y', \mathbf{x})}$$

To adapt logistic regression to the ranking objective, we need to adjust the normalization term to sum only over those assignments generated by $q(\mathbf{y}|\mathbf{x})$:

$$p(y|\mathbf{x}, K(q, \mathbf{x})) = \frac{\exp \sum_i \lambda_i \phi_i(y, \mathbf{x})}{\sum_{y' \in K(q,\mathbf{x})} \exp \sum_i \lambda_i \phi_i(y', \mathbf{x})}$$

Given a set of ranking training examples, the parameters of this model can be estimated using standard numerical optimization methods such as BFGS [63].

While Collins [16] has reported positive results using a reranking classifier to improve the output of a syntactic parser, the main drawback of reranking estimation is that it is limited by the quality of the top K assignments generated by $q(\mathbf{y}|\mathbf{x})$. In many tasks, the space of possible outputs is so large that K has to be very large to contain enough diversity for improved accuracy.

5.2 SampleRank Estimation

The main computational problem with the online learning methods described above is that they all require the prediction algorithm to run to completion before parameters are

updated. For many real-world problems, prediction is very computationally intensive, so delaying updates this long may be inadvisable. Furthermore, given the fact that we will be using MCMC algorithms for inference, it is desirable to have a learning algorithm that is well-suited for sampling algorithms.

In this section, we present a general learning scheme we call *SampleRank* (Algorithm 4). Like the previous online learning methods, parameter updates are made when the prediction algorithm makes an error on the training examples. However, in SampleRank, the definition of an error is customized to the fact that a sampling algorithm is used for inference. In SampleRank, an error occurs when the sampling algorithm misranks a pair of samples. Therefore, SampleRank potentially updates parameters after each sample is drawn. This difference provides two advantages: (1) by training on more pairs of assignments, SampleRank leverages a more comprehensive set of training examples, (2) by updating parameters after each sample, SampleRank can more rapidly find a good set of parameters.

The general version of SampleRank is presented in Algorithm 4. The algorithm iterates over the training data using the sampling algorithm S to generate assignments $\mathbf{y}$. After each sample $\mathbf{y}'$ is generated, a call is made to UPDATEPARAMETERS, which updates the parameters Λ if an error is made. We discuss different forms of the update computation in the Section 5.2.1. After an update is made, a new state must be chosen. In Section 5.2.2, we outline two possible implementations of this method.

5.2.1 Parameter Updates

There are three decisions we must make to update parameters:

- **Determining whether to perform an update:** In traditional online learning, an update is performed if an error exists between the prediction and the ground truth. We need to adapt the definition of an *error* to the case of *SampleRank*.

Algorithm 4 SampleRank Estimation

Input: training data $\mathcal{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}) \dots (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$
initial parameters Λ
sampling algorithm $S(\mathbf{y}, \mathbf{x}, \Lambda) \mapsto \mathbf{y}'$
Number of samples per instance N
for $t \leftarrow 1$ to number of iterations T **do**
  for training example $(\mathbf{y}^{(j)}, \mathbf{x}^{(j)}) \in \mathcal{D}$ **do**
    Sample initial state $\mathbf{y}$
    for $s \leftarrow 1$ to N **do**
      generate sample $\mathbf{y}' = S(\mathbf{y}, \mathbf{x}^{(i)}, \Lambda)$
      $\Lambda \leftarrow$ UPDATEPARAMETERS$(\mathbf{y}, \mathbf{y}', \Lambda)$
      $\mathbf{y} \leftarrow$ CHOOSENEXTSTATE$(\mathbf{y}, \mathbf{y}')$
    end for
  end for
end for

- **Determining what assignment to update toward:** In traditional online learning, the "positive" example in each update is the ground truth. Here, we extend this notion to also allow updates toward imperfect assignments.

- **Determining the functional form of the update:** We propose three forms of the update equations.

The first notion we need to define for the parameter update is exactly what an error is. Let $g_\Lambda(\mathbf{y}) = \prod_i f_i(\mathbf{y}, \mathbf{x})$ be the unnormalized probability of assignment $\mathbf{y}$ according to the model with parameters Λ. We will also refer to this unnormalized probability as the *score* for an assignment. Let $L(\mathbf{y}, \mathbf{y}^*)$ be the *loss* of the proposed assignment $\mathbf{y}$ compared to the correct solution $\mathbf{y}^*$. For example, the loss can be the inverse of accuracy.

Given a pair of samples $(\mathbf{y}, \mathbf{y}')$, we say the samples are *in error* if the model assigns a higher score to the sample with the lower loss, i.e.:

$$[(g_\Lambda(\mathbf{y}) > g_\Lambda(\mathbf{y}')) \wedge (L(\mathbf{y}, \mathbf{y}^*) > L(\mathbf{y}', \mathbf{y}^*))] \vee$$
$$[(g_\Lambda(\mathbf{y}) < g_\Lambda(\mathbf{y}')) \wedge (L(\mathbf{y}, \mathbf{y}^*) < L(\mathbf{y}', \mathbf{y}^*))]$$

Next, we specify how to generate an assignment to update toward, which we refer to as the *target assignment*. In traditional online learning, the *target assignment* is simply the ground truth ($\mathbf{y}^*$). However, here we will consider additional target assignments to enforce the correct ranking of incorrect assignments. First, we describe how to generate *neighboring* assignments.

Let $\eta(\mathbf{y})$ be a *neighborhood* function on $\mathbf{y}$, i.e., $\eta : \mathbf{y} \mapsto \{\mathbf{y}_1 \dots \mathbf{y}_k\}$. The form of the neighborhood function depends on the underlying sampling algorithm. For example, in Gibbs sampling, $\eta(\mathbf{y})$ is the set of all possible assignments to the variable Y_i that is to be resampled. If we let $D(Y_i)$ be the set of possible assignments to Y_i, then this neighborhood function is defined as:

$$\eta_G(\mathbf{y}) = \{\mathbf{y}' : \mathbf{y} \setminus y_i = \mathbf{y}' \setminus y_i\} \quad \forall y_i \in D(Y_i)$$

For Metropolis-Hastings, $\eta(\mathbf{y})$ is the set of all assignments having non-zero probability according to the proposal distribution:

$$\eta_{MH}(\mathbf{y}) = \{\mathbf{y}' : q(\mathbf{y}'|\mathbf{y}) > 0\}$$

Let $\mathbf{y}$ be the element of the sample pair $(\mathbf{y}, \mathbf{y}^*)$ having the largest loss. We distinguish between two methods of selecting the target assignment from the neighborhood of $\mathbf{y}$: *best neighbor* and *closest better neighbor*. *Best neighbor* selects the element of $\eta(\mathbf{y})$ that has the smallest loss:

$$\text{BestNeighbor}(\mathbf{y}) = \underset{\mathbf{y}' \in \eta(\mathbf{y})}{\operatorname{argmin}} L(\mathbf{y}', \mathbf{y}^*)$$

The motivation behind *best neighbor* is to update toward the best possible sample that could have been generated at the current time step.

ClosestBetterNeighbor selects the element of $\eta(\mathbf{y})$ that has a smaller loss than $\mathbf{y}$ and also has the highest model score according to g_Λ:

$$\begin{aligned}\text{ClosestBetterNeighbor}(\mathbf{y}) \;=\; & \operatorname*{argmin}_{\mathbf{y}' \in \eta(\mathbf{y})} g_\Lambda(\mathbf{y}) - g_\Lambda(\mathbf{y}') \\ & \text{s.t.} \\ & g_\Lambda(\mathbf{y}') < g_\Lambda(\mathbf{y}) \wedge L(\mathbf{y}, \mathbf{y}^*) > L(\mathbf{y}', \mathbf{y}^*)\end{aligned}$$

The motivation for *closest better neighbor* is to minimize parameter fluctuations by updating toward examples that have similar model scores. Additionally, this focuses updates on small differences between samples. For both *best neighbor* and *closest better neighbor*, ties are broken randomly.

Finally, we consider three functional forms of the update. Let $\mathbf{y}^+$ be the target assignment for sample $\mathbf{y}$ (obtained either by the *best neighbor* or *closest better neighbor* method). Then the updates are as follows:

- **Perceptron:** The perceptron update from line 6 of Algorithm 2. Adapted here, the update is

$$\Lambda^{t+1} = \Lambda^t + \big(\Phi(\mathbf{y}^+, \mathbf{x}) - \Phi(\mathbf{y}, \mathbf{x})\big)$$

- **MIRA:** The MIRA update adapted from Equation 5.2:

$$\Lambda^{t+1} = \operatorname*{argmin}_{\Lambda} ||\Lambda^t - \Lambda||^2 \;\; s.t. \;\; g_\Lambda(\mathbf{y}^+) - g_\Lambda(\mathbf{y}) \geq M$$

- **MIRA++:** This is a slight variation of the MIRA update motivated by the fact that using the MIRA update alone, the learning algorithm may rarely make updates toward the overall true assignment. Thus, the optimal assignment in the training example is not guaranteed to have the highest model score. We augment the previous MIRA

update with two additional constraints ensuring that the optimal assignment $\mathbf{y}^*$ is ranked above both $\mathbf{y}^+$ and $\mathbf{y}$ (assuming $\mathbf{y}^+ \neq \mathbf{y}^*$):

$$\Lambda^{t+1} = \underset{\Lambda}{\operatorname{argmin}} ||\Lambda^t - \Lambda||^2 \;\; s.t. \;\; g_\Lambda(\mathbf{y}^+) - g_\Lambda(\mathbf{y}) \geq M,$$
$$g_\Lambda(\mathbf{y}^*) - g_\Lambda(\mathbf{y}^+) > M,$$
$$g_\Lambda(\mathbf{y}^*) - g_\Lambda(\mathbf{y}) > M$$

Although no longer computable in closed form, this update can be calculated quickly using Hildreth and D'esopo method [14]. For both MIRA and MIRA++ updates, we set the margin $M = 1$. Although elsewhere the margin has been set to the loss function for the assignments [69], in initial experiments we did not find that this addition affected the accuracy of results.

Note that since all of these updates operate only on pairs of assignments, we benefit from the memory-efficiency described in Section 4.3. Namely, we can avoid instantiating all variables whose setting is the same for both assignments.

5.2.2 Sample to Accept

Once we have updated the parameters, we must next decide how to proceed with sampling on the current training example. We consider two possibilities:

- **Predicted**: Select the sample $\mathbf{y}'$ that was generated by the sampling algorithm, *even if* $\mathbf{y}'$ *is worse than the previous assignment* $\mathbf{y}$.
- **Truth**: Select the sample $\mathbf{y}^+$ that was chosen as the positive example for the parameter update.

After initial experimentation, **Predicted** consistently outperformed **Truth**, so we only use **Predicted** in the experiments below. This behavior is likely because using only true

Configuration	Possible Settings
Sampling Algorithm	Gibbs, Metropolis-Hastings
Temperature	Fixed real value, cooling schedule
Target Assignment	best neighbor, closest improved neighbor
Update type	Perceptron, MIRA, MIRA++
Sample to Accept	predicted, truth
Samples per example	The number of samples (search steps) per example
Iterations	The number of training iterations over all examples

Table 5.1. A summary of the choices required to instantiate a version of the SampleRank algorithm.

samples quickly leads to the optimal solution, generating few suboptimal training examples. (Although, see the Searn algorithm [27] for an example of interpolating between these two extremes.)

Table 5.1 summarizes the various possible instantiations of the SampleRank algorithm. In Chapters 6 - 8 we empirically investigate the quality of various SampleRank configurations.

5.3 Related Work

There has been a recent interest in training methods that enable the use of flexible representations. Perhaps the most related are "Learning as Search Optimization" (LaSO) and Searn [26, 25]. Like our proposed approach, LaSO and Searn are also error-driven training methods that integrate prediction and training. However, whereas we explicitly use a ranking-based loss function, LaSO and Searn use binary classification loss functions that label each search action as *correct* or *incorrect*. A major difference between SampleRank and Searn is that in SampleRank we compute features over an *entire solution*, whereas Searn only computes features over a *search action*. Thus, it is not obvious how to convert Searn into a probabilistic inference algorithm. In SampleRank, however, we can obtain probabilities by normalizing the score for a solution by the scores for all other solutions (which we can approximate through sampling). Another difference is that SampleRank is

an online algorithm, while Searn updates parameters only after seeing all training examples (and furthermore requires a development set to choose the step size of each update).

A similar technique is the "pipeline" approach to structured classification, which learns a classifier to make local decisions in a search algorithm [92, 116, 15]. Like Searn, this approach does not compute model scores for a complete assignment, only for local search actions.

Collins and Roark [18] present an incremental perceptron algorithm for parsing that uses "early update" to update the parameters when an error is encountered. Our method uses a similar "early update" in that parameters are modified when an error is made during sampling. However, SampleRank differs both in the form of the updates (MIRA) and the assignment we update toward (the better of two samples, not the truth).

Recently, Lowd and Domingos [64] compared a number of learning algorithms for Markov logic networks, and determined that ill-conditioning can be a serious problem in parameter estimation. They adopt algorithms to address this by using different learning weights for different parameters. It is possible that adopting the updates in SampleRank to address ill-condition will also improve performance.

There have also been a number of reinforcement learning techniques proposed to improve combinatorial optimization. While most operate over completely instantiated solutions [120, 9, 74], Bertsekas and Tsitsiklis [4] do suggest ways to learn how to construct a solution, however it is not clear if this approach is scalable.

Others have attempted to train with flexible representations using Gibbs sampling [35], message propagation, [13, 104], and integer linear programming [96]. While many of these related approaches require a careful factorization to enable efficient inference, here we can use the full flexibility of weighted logic.

The main distinctions of our approach are that it is simple to implement, not computationally intensive, and adaptable to arbitrary loss functions.

Sutton and McCallum [106] present a related learning approximation called *piecewise estimation*. Piecewise training avoids intractable normalization constants by splitting a factor graph into independent "pieces" and training classifiers to predict each piece independently. At test time, joint inference is performed across all pieces. Whereas piecewise estimation ignores the biases of inference during training time, SampleRank fully integrates learning and prediction. We believe this integrated learning can result in parameters that are better suited for the prediction algorithm.

CHAPTER 6

SYNTHETIC EXPERIMENTS

In this section, we present results with synthetic data to determine under what conditions we can expect SampleRank to perform well.

Through the use of weighted logic, we are sacrificing exact learning and inference algorithms for more advanced knowledge representations. These advanced representations can model dependencies that simpler representations cannot. This trade-off will only be beneficial if these more complex dependencies are actually predictive of good solutions. We refer to these more complex dependencies as *non-local representations*, in contrast to the *local* representations typically used to enable dynamic programming. In the following experiments, we explore the following hypothesis: **As non-local representations become more predictive, we expect the comparative advantage of weighted logic solutions to increase.**

6.1 Data Generation

To test the above hypothesis, we require a domain for which exact inference and learning is tractable with non-local dependencies. We therefore turn to linear-chain sequence models. To generate synthetic data, we sample from a second-order Markov chain, an example of which is shown in Figure 6.1. Here, $\mathbf{y}$ are unobserved discrete variables with four possible assignments each, $\mathbf{x}$ are observed discrete variables with 20 possible assignments each. Each sampled sequence contains 20 y variables and 20 x variables.

We generate the parameters of the synthetic model randomly from a Gaussian with mean 1. To control the impact of the second-order dependencies, we alter the variance

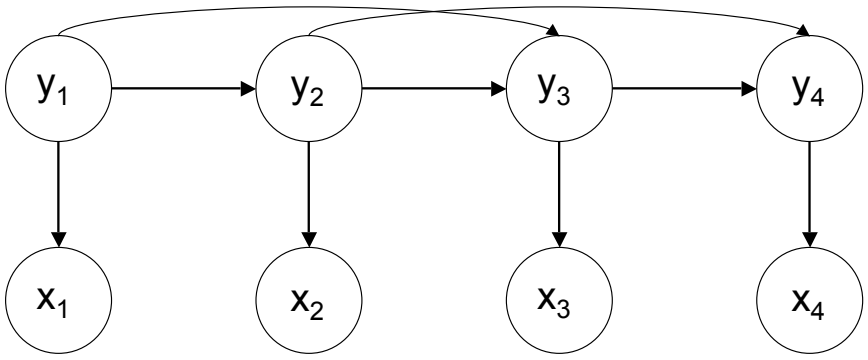

Figure 6.1. 2nd-order Markov model used to generate synthetic data.

of the parameters for second-order factors (e.g., y_1 and y_3 in Figure 6.1). This allows us to determine how much of an impact the second-order features have on the generate data, and thus how *predictive* the non-local dependencies are. We refer to the variance of these second-order parameters as σ^2. As σ^2 increases, the second-order dependencies become more predictive, and thus a second-order CRF becomes a better model than a first-order CRF.

6.2 Systems

We estimate three models from this data:

- **CRF1**: A first-order CRF. This represents the "simple" representation that does not capture the non-local (i.e., second-order) dependencies.

- **CRF2**: A second-order CRF. This represents the "non-local" representation; however, because of the simplicity of the model, we can compute exact solutions to the learning and inference problems.

- **SampleRank-M++**: The SampleRank algorithm proposed in the previous chapter using the second-order dependencies with the following configuration: MIRA++ parameter update, Gibbs sampling with fixed temperature 0 (i.e., coordinate ascent),

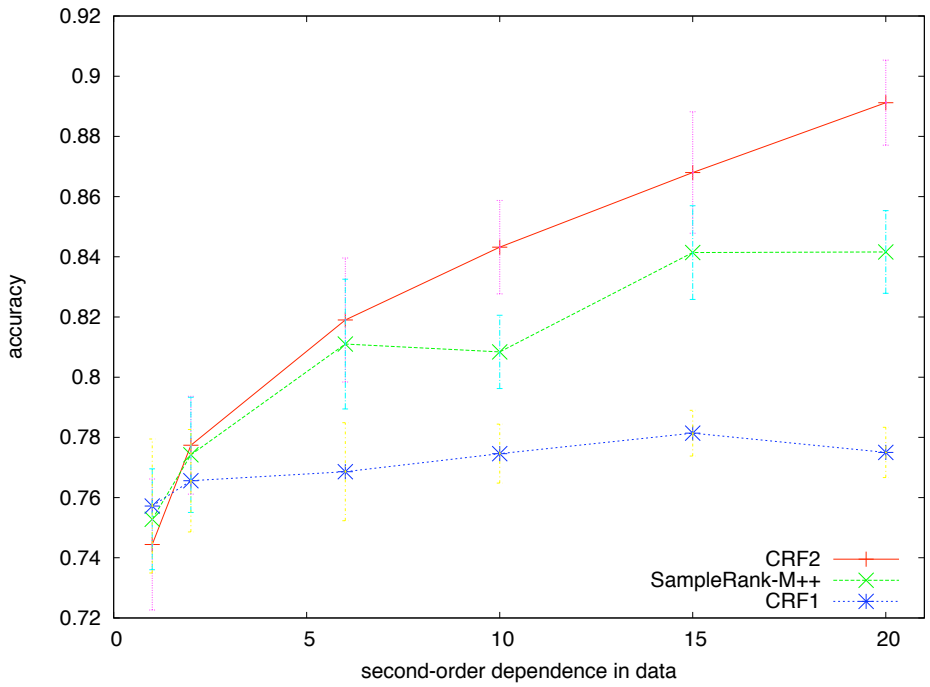

Figure 6.2. Results on synthetic data. As non-local dependencies become more predictive, it become more important to model them. Further, the approximation of SampleRank is quite good for lower values of second-order dependencies, but becomes less exact as these dependencies become more deterministic.

best neighbor updating, 50 samples per instance, and 10 iterations over the training data.

Thus, the difference between **CRF1** and **CRF2** is the upper-bound on the possible improvement to be gained from the non-local representation. The difference between **second-order** and **SampleRank-M++** is the difference between using exact and approximate learning and inference algorithms.

6.3 Results

Figure 6.2 displays the results over 5 random trials of the experiment. All differences at $\sigma^2 = 20$ are statistically significant at the 0.05 confidence level using McNemar's test. These results indicate that as the non-local dependencies become more predictive, models that capture these dependencies outperform those that do not by a wider and wider

margin. Furthermore, we can observe that the approximation of SampleRank appears to degrade slightly as the dependencies become stronger. That is, the difference between **SampleRank-M++** and **CRF2** increases as the non-local dependencies increase. This suggests the difficulty of learning near-deterministic dependencies. That is, as σ^2 grows, the second-order dependencies become stronger because the sampled parameter values become larger.

Notice also that when the second-order dependence is low, the results from each model do not differ in a statistically meaningful way. This suggests that it is not worth the effort to use SampleRank if the local dependencies are just as predictive as the non-local dependencies. Although, it is encouraging that SampleRank does not do noticeably worse than the simpler model in this case.

We next consider how well different approximation algorithms perform. In Figure 6.3, we additionally compare:

- **SampleRank-M**: SampleRank using the original MIRA update (i.e., without the extra constraint for the true assignment). Otherwise, the same configuration is used as in **SampleRank-M++**.

- **MIRA**: Standard MIRA training (i.e., not embedded in SampleRank). Updates are only made between the final predicted solution and the truth. As in SampleRank, we run MIRA for 10 iterations over the data.

Somewhat surprisingly, MIRA performs quite poorly on its own, and improves dramatically when embedded within SampleRank. We attribute this to the fact that when used within SampleRank, MIRA obtains many more "training instances" in the form of pairs of assignments, whereas on its own it only receives two assignments per instance (the predicted and the truth). In the next section, we will see this effect more clearly by comparing accuracies of **MIRA** and **SampleRank-M** as the number of training iterations varies.

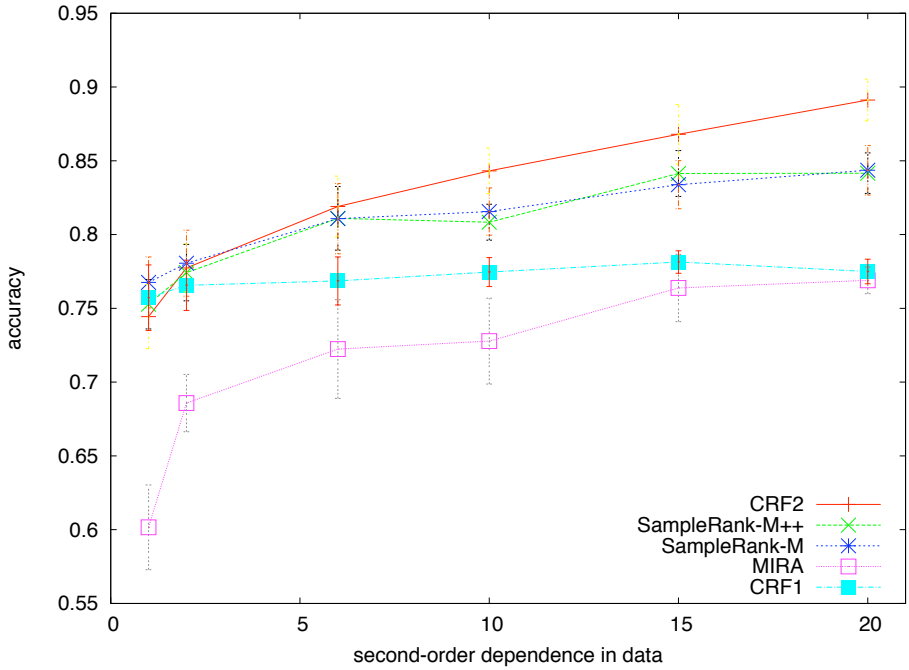

Figure 6.3. Results on synthetic data comparing different uses of the MIRA parameter update. In general, MIRA alone performs quite poorly, but when embedded in the SampleRank algorithm, accuracy improves significantly.

At $\sigma^2 = 20$, the differences between **SampleRank-M** and **SampleRank-M++** is not statistically significant, suggesting that the additional constraint from the true assignment is not needed for this data. Also, the and between **CRF1** and **MIRA** is not statistically significant (McNemar's test, $\alpha = 0.05$).

In Figure 6.4, we perform an analogous comparison using the Perceptron algorithm, where again we observe that embedding the online learner within the SampleRank algorithm significantly improves performance. Again, we credit this to the increased number of ranking training examples generated by the SampleRank algorithm. Notice also that Perceptron alone appears to outperform MIRA alone. (All differences are statistically significant using McNemar's test, $\alpha = 0.05$).

Finally, we also compare with a reranking classifier. This is a logistic-regression classifier that uses the same features as the non-local representation used in the SampleRank, Perceptron, and MIRA experiments above. The classifier takes as input the top k predic-

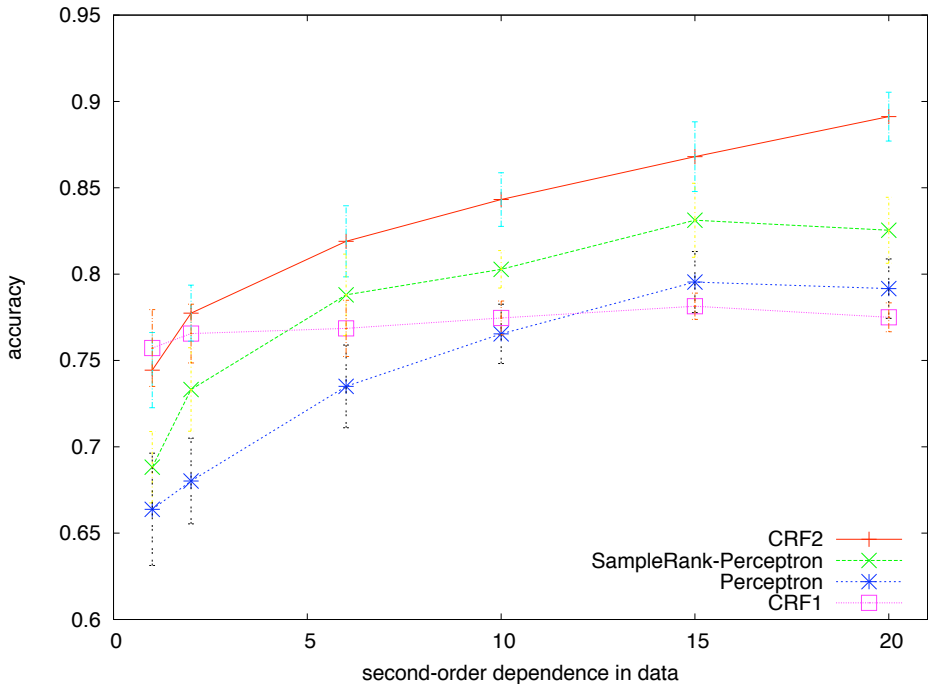

Figure 6.4. Results on synthetic data comparing different uses of the Perceptron parameter update. As in Figure 6.3, the online learner alone performs quite poorly when compared to exact learning, but when embedded in the SampleRank algorithm, accuracy improves significantly.

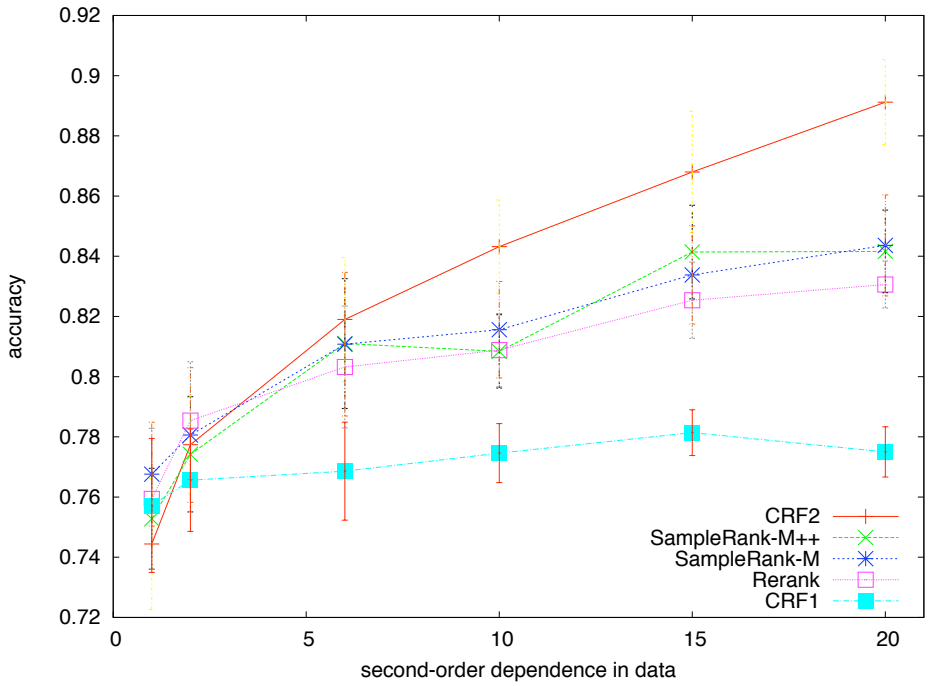

Figure 6.5. Results on synthetic data comparing Reranking with SampleRank using MIRA and MIRA++ updates.

tions of **CRF1** and reranks them using the features over second-order dependencies. For a comparison with the SampleRank algorithm, we set k to 50 (because SampleRank uses 50 samples per instance).

Figure 6.5 shows the results of reranking compared to **SampleRank-M** and **SampleRank-M++**. Reranking actually does quite well on this data, and is competitive with the SampleRank algorithms. Both SampleRank algorithms significantly better than **Rerank** at $\sigma^2 = 20$, though the results differences are not significant for smaller values of σ^2 (McNemar's, $\alpha = 0.05$). We believe reranking performs well here because the state space is relatively small. (The output variable dimension is 4.) This fact means that the top 50 predictions of **CRF1** may actually often contain the true assignment. In the next section, we will present results showing that reranking performs quite poorly when this state space is larger (dimension 26).

CHAPTER 7

NAMED ENTITY RECOGNITION EXPERIMENTS

We now turn to experiments on real problems in NLP. As described in Chapter 3, named entity recognition (NER) is the task of labeling word tokens with labels indicating the type of named-entity they refer to, for example person, place, or organization. Typically this is modeled using a linear-chain sequence model (CRF or HMM). Because these models often make a first-order Markov assumption among the output variables, only the compatibility between adjacent output labels is measured. Consequently, models with such simple representations can make embarrassing mistakes. There have been examples of extensions to these linear-chain models to capture specific dependencies. For example, Sutton and McCallum [104] propose the "skip-chain" feature that indicates whether all mentions of a named entity in a document are assigned the same label. However, we would like to avoid constructing new models with each non-local feature, and instead employ weighted logic to represent dependencies.

7.1 Data

In this section, we present NER experiments on the problem of extracting information from research paper citations. Given a set of citations, the task is to extract the following entities: author, title, journal, booktitle, tech-report, volume, number, date, editor, institution, location, publisher, and notes (e.g., "under review"). Figure 7.1 shows an example.

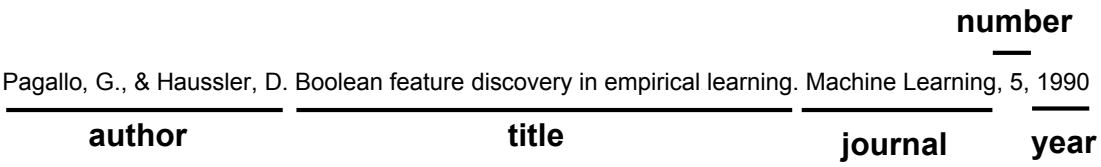

Figure 7.1. An example citation from the Cora dataset.

We obtain 500 citations from the "tagged references" component of the Cora IE dataset [1]. We perform three-fold cross-validation for all experiments.

We use the following features for all models:

- The token itself.
- Regular expressions indicating whether a token contains punctuation, capitalization, digits.
- The same features for each token to the right and left of the current token (e.g., "the previous token is capitalized").

We also include non-local features for all models except the linear-chain CRF (which cannot represent these features):

- If a *journal* label is predicted, indicate whether a *volume* label is also predicted.
- If a *tech-report* label is predicted, indicate whether an *institution* label is also predicted.
- Does the prediction contain a title? A date? An author? A venue?
- Is any entity type duplicated? (e.g. multiple paper titles)
- The number of non-numeric tokens in a volume, date, or journal number field.
- The number of extracted fields that only contain punctuation.

[1] `www.cs.umass.edu/mccallum/data/cora-ie.tar.gz`

These non-local features attempt to capture some of the complex constraints in the data that cannot be modeled by a linear-chain CRF. While some of these features could be captured by a semi-Markov CRF or by a grammar-based model, the full flexibility of weighted logic makes it easy to incorporate all of these features without worrying about the factorization of the model.

7.2 Systems

While linear-chain CRFs have performed quite well on this task [85], we are interested in whether the additional features enabled by weighted logic would result in more accurate models. We therefore compare the following models: SampleRank-M++, SampleRank-M, SampleRank-Perceptron, CRF (with a first-order Markov assumption), Perceptron, Mira, and Reranking. For all the sampling approaches, we use the greedy setting of Gibbs sampling (i.e., coordinate ascent), best neighbor updating, 50 samples per instance, and 70 iterations over the training data. For the reranking approach, we use a beam of size 50 to make the results comparable with SampleRank.

7.3 Results

Figure 7.2 shows the results averaged over three cross-validation splits of the data. Using McNemar's test for the results after 70 iterations, all differences are statistically significant at the 0.05 confidence level except for the difference between **SampleRank-M** and **CRF**.

First, we can observe that SampleRank-M++ outperforms all competing models. In particular, the additional linear constraint in the MIRA++ update appears to make a significant difference in these results. Second, we can observe that traditional online learning methods were unable to outperform the simple CRF. This suggests that the quality of the learning algorithm is a critical decision when incorporating non-local features.

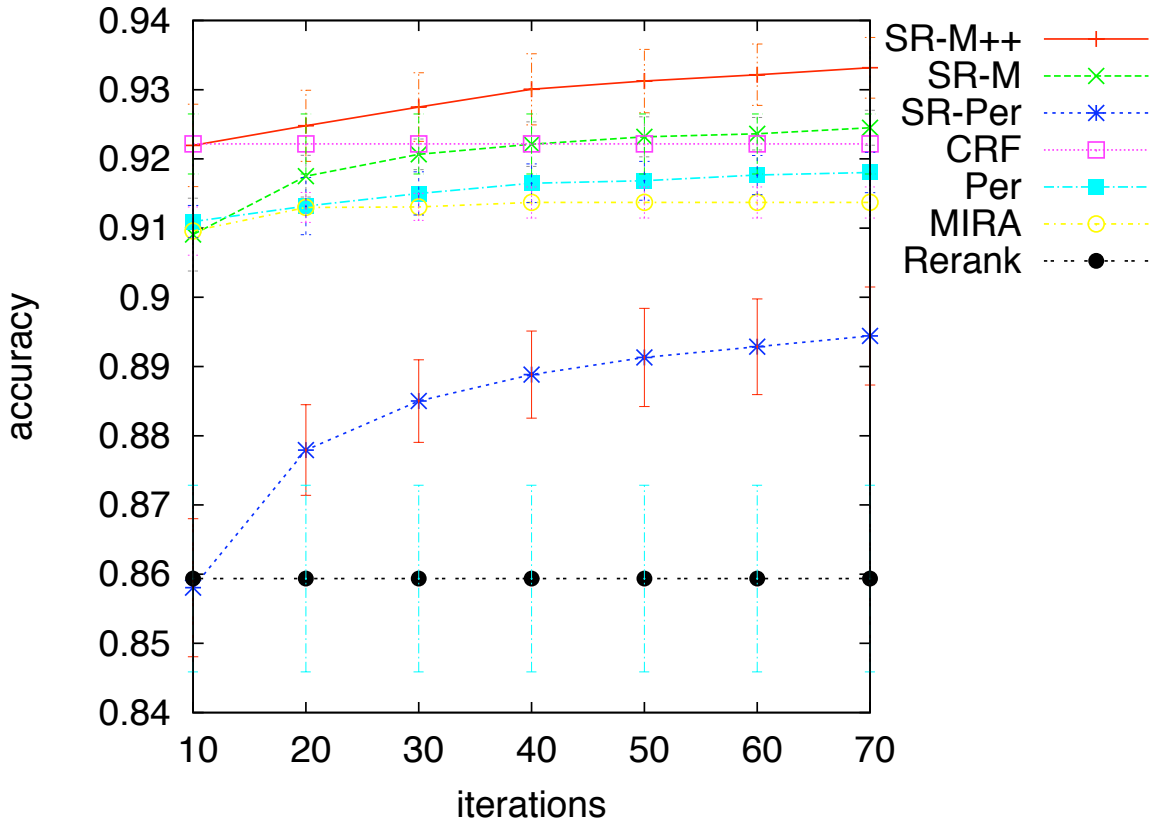

Figure 7.2. Results on named-entity recognition on the Cora citation dataset.

Examining the output, we notice that often the non-local features help capture global constraints over the output. For example, consider the following citation:

> *Allender , N . Immerman , J . Balcazar (1993) , A First Order Isomorphism Theorem , STACS , 163 - 174*

The CRF incorrectly labels *A First Order Isomorphism Theorem, STACS* as a conference. (The CRF may be confused because the token *first* often appears in conference names, e.g., *First Conference on Artificial Intelligence*.) Because there is a non-local feature indicating that the prediction contains no title entity, the **SampleRank-M++** approach correctly segments the title and conference for this paper.

A few comparative results differ from those in the synthetic experiments. First, Reranking performs quite poorly. We attribute this to the much larger state space in this real-world problem. We adopted the common "B-I-O" labeling convention to indicate the beginning and inside of entities (e.g., *B-author*). This led to a total of 26 possible output labels. With an average citation length of 37 tokens, this leads to a very large number of possible label sequences. The top 50 predictions of the first-order CRF may simply not be diverse enough.

Another variation from the synthetic results is that Perceptron actually performs worse when embedded within the SampleRank algorithm. Because MIRA does perform better when part of SampleRank, we attribute the failure of Perceptron to the large parameter fluctuations caused by the form of its update. Figure 7.3 plots the number of parameter updates made at each round of training. As we can see, SampleRank-Perceptron continues to make a large number of updates even after many rounds of training, while SampleRank-MIRA seems to be converging more steadily. This is somewhat expected, since the norm minimization in the MIRA update is designed to reduce such fluctuations.

Finally, to investigate how important the non-local features are to the success of SampleRank, we also ran a version of SampleRank-M++ using only the base features used by the CRF. This resulted in a very low accuracy of 78.2, significantly less than the original

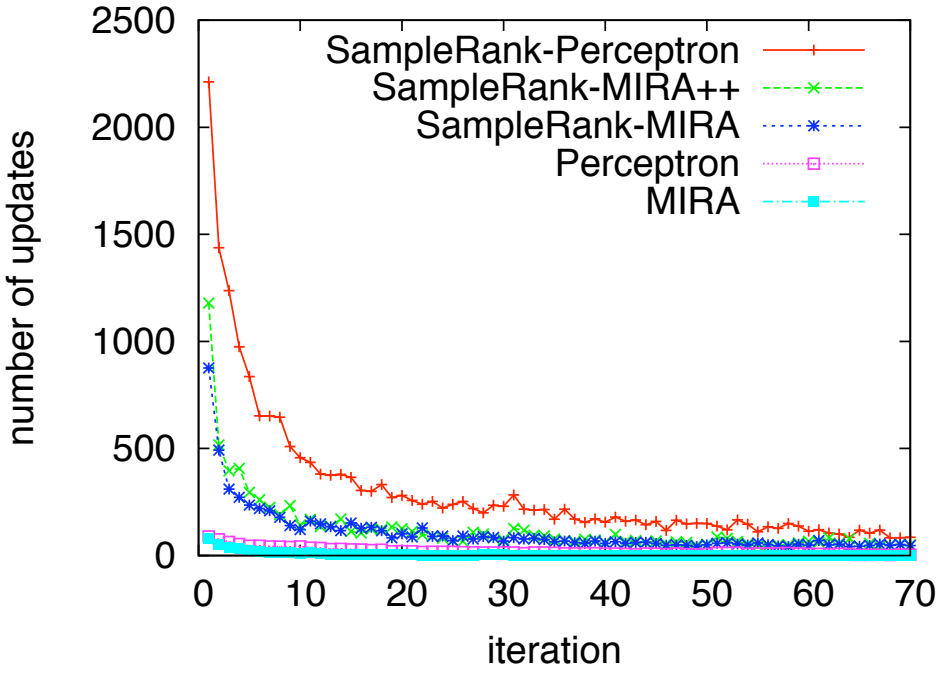

Figure 7.3. The number of parameter updates made at each iteration for the various online learning algorithms on the Cora NER data. This figure indicates that SampleRank-Perceptron may be performing poorly because of the large fluctuations in parameters that is exacerbated by updating after each sample.

CRF accuracy of 92.2. This highlights the limitations of approximate learning and inference algorithms and suggests that these limitations can be overcome with a rich enough representation. Indeed, examining the parameter values for features in the SampleRank models, we observe that the features with the highest absolute weights are the non-local features not present in the linear-chain CRF. Example of features with high absolute weights include whether the volume field contains only numeric tokens (positive), whether there exists a field with only punctuation (negative), and whether the output has more than one date field (negative).

Finally, as a point of reference, inter-annotator agreement for similar NER tasks is typically in the high 90s [101]

CHAPTER 8

NOUN PHRASE COREFERENCE RESOLUTION EXPERIMENTS

8.1 Task and Data

In this section, we apply SampleRank to the task of coreference resolution of noun phrases. The input is a document annotated with the location of entity mentions such as *Hillary Clinton ... he ... Bill Clinton ... the senator ... he*. The output is a disjoint clustering of mentions into coreferent sets such as {*Hillary Clinton, the senator*}, {*he, Bill Clinton, he*}.

Traditional machine learning approaches to coreference resolution use a simple propositional representation which reduces to learning a binary classifier over *pairs* of mentions [100, 80]. The evidence variables therefore represent textual and syntactic similarities between pairs of mentions. For example, features for the mention pair (*Bill Clinton, he*) might include (*GendersMatch, InSameSentence*).

This pairwise factoring prevents the evidence variables from representing information about sets of mentions or from reasoning about aggregate properties of entities. We instead propose using a weighted logic representation over an entire solution, i.e., an entire clustering of mentions. This enables cluster-wide evidence, for example indicating whether a cluster only contains pronominal mentions. Since most personal pronouns have an antecedent earlier in the document, the weight for this feature should penalize this prediction. We discuss these features more in Section 8.1.1.

The difference between the pairwise representation and the weighted logic representation can be understood in terms of their corresponding factor graphs. The pairwise factor graph has a binary query variable y for each pair of mentions. In contrast, the weighted

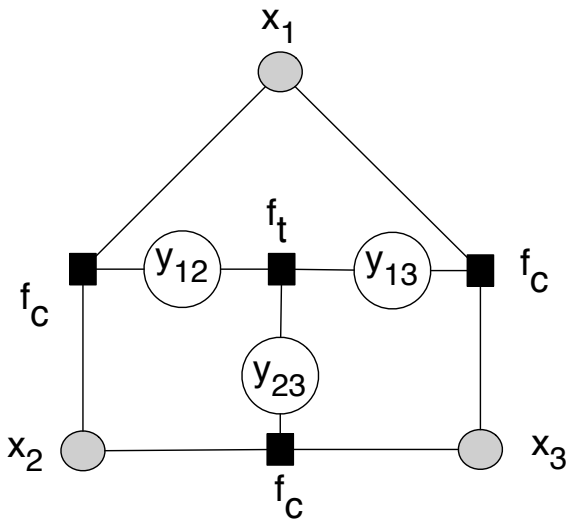

Figure 8.1. An example noun coreference factor graph for the Pairwise Model in which factors f_c model the coreference between two nouns, and f_t enforce the transitivity among related decisions. The number of y variables increases quadratically in the number of x variables.

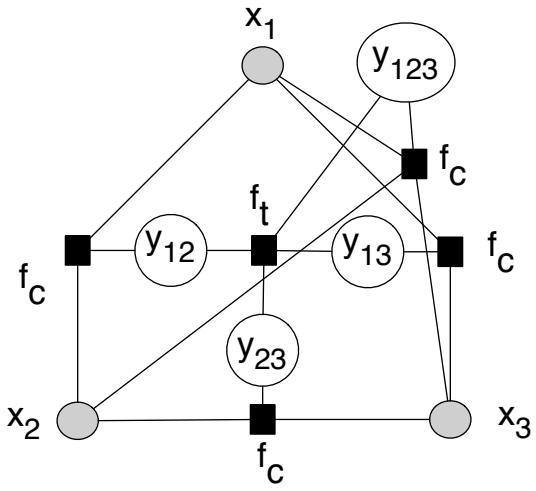

Figure 8.2. An example noun coreference factor graph for the weighted logic model in which factors f_c model the coreference between *sets* of nouns, and f_t enforce the transitivity among related decisions. Here, the additional node y_{123} indicates whether nouns $\{x_1, x_2, x_3\}$ are all coreferent. The number of y variables increases exponentially in the number of x variables.

logic factor graph has a binary query variable for *every subset of mentions.* Clearly, instantiating all such variables explicitly is infeasible. Figures 8.1 and 8.2 show factor graphs for the pairwise and first-order models for the case of three mentions. Here, y variables correspond to query variables, and shaded x variables correspond to evidence variables.

Note that even with the pairwise propositional representation exact dynamic programming is not possible for coreference resolution. This is the case because the pairwise coreference predictions are not made independently; rather, the transitivity among the decisions forces the prediction algorithm to score many related decisions simultaneously.[1] Previous systems have performed prediction using agglomerative clustering or graph partitioning [100, 80, 68].

[1] If (A, B) and (B,C) are coreferent, then so is (A,C).

We apply our approach to the noun coreference ACE 2004 data[2], containing 443 news documents with 28,135 noun phrases to be coreferenced. 336 documents are used for training, and the remainder for testing. All entity types are candidates for coreference (pronouns, named entities, and nominal entities). We use the true entity segmentation, and parse each sentence in the corpus using a phrase-structure grammar, as is common for this task.

8.1.1 Features

We follow Soon et al. [100] and Ng and Cardie [80] to generate most of the evidence variables for the Pairwise representation. These include:

- Match features - Check whether gender, number, head text, or entire phrase matches
- Mention type (pronoun, name, nominal)
- Aliases - Heuristically decide if one noun is the acronym of the other
- Apposition - Heuristically decide if one noun is in apposition to the other
- Relative Pronoun - Heuristically decide if one noun is a relative pronoun referring to the other.
- Wordnet features - Use Wordnet to decide if one noun is a hypernym, synonym, or antonym of another, or if they share a hypernym.
- Both speak - True if both contain an adjacent context word that is a synonym of "said." This is a domain-specific feature that helps for many newswire articles.
- Modifiers Match - for example, in the phrase "President Clinton", "President" is a modifier of "Clinton". This feature indicates if one noun is a modifier of the other, or they share a modifier.

[2]http://www.nist.gov/speech/tests/ace/

- Substring - True if one noun is a substring of the other (e.g. “Egypt” and “Egyptian”).

The First-Order representation includes the following variables[3]:

- Enumerate each pair of noun phrases and compute the features listed above. **All-X** is true if all pairs share a feature X, **Most-True-X** is true if the majority of pairs share a feature X, and **Most-False-X** is true if most of the pairs do not share feature X.

- Use the output of the Pairwise Model for each pair of nouns. **All-True** is true if all pairs are predicted to be coreferent, **Most-True** is true if most pairs are predicted to be coreferent, and **Most-False** is true if most pairs are predicted to not be coreferent. Additionally, **Max-True** is true if the maximum pairwise score is above threshold, and **Min-True** if the minimum pairwise score is above threshold.

- Cluster Size indicates the size of the cluster.

- Count how many phrases in the cluster are of each mention type (name, pronoun, nominal), number (singular/plural) and gender (male/female). The features **All-X** and **Most-True-X** indicate how frequent each feature is in the cluster. This feature can capture the soft constraint such that no cluster consists only of pronouns.

In addition to the listed features, we also include conjunctions of size 2, for example “Genders match AND numbers match”.

8.2 Systems

In these experiments, the prediction algorithm is greedy Metropolis-Hastings with a proposal distribution that only considers merging clusters, i.e., greedy agglomerative clustering. A solution is a clustering of mentions, and a neighboring solution is the set of all

[3]Note that some of these evidence variables require extensions of standard first-order logic, e.g., to compute the average of a set of constants.

possible clusterings formed by merging existing clusters. The parameters are therefore learned to rank possible improvements to existing clusterings.

We compare systems along two dimensions, *representation* and *learning*. We consider two types of representation:

- **Weighted logic**: The representation advocated above, where evidence variables are computed over sets of mentions.
- **Pairwise**: The baseline mentioned above, where evidence is only computed over pairs of mentions.

We consider two types of learning:

- **SampleRank-M**: The SampleRank algorithm with MIRA parameter updates. In these experiments, we update toward the *closest improved neighbor*, accept predicted samples only, and make 30 passes over the training data. Rather than using a fixed number of samples per example, we perform agglomerative clustering until no more clusters can be merged and improve the model score.
- **Binary Classification**: For the pairwise representation, this corresponds to generating binary training examples from all pairs of mentions and learning a binary logistic regression classifier. For the first-order representation, we sample sets of positive and negative examples of clusters and learn a binary logistic regression classifier to predict whether an entire set of mentions is coreferent.

8.3 Results

We use the B^3 algorithm to evaluate the predicted coreferent clusters [2]. B^3 is common in coreference evaluation and is similar to the precision and recall of coreferent links, except that systems are rewarded for singleton clusters. For each noun phrase x_i, let c_i be the number of mentions in x_i's predicted cluster that are in fact coreferent with x_i (including

	F1	Prec	Rec
Weighted Logic SampleRank-M	**79.3**	86.7	73.2
Pairwise SampleRank-M	72.5	92.0	59.8
Weighted Logic Classification	**69.2**	79.0	61.5
Pairwise Classification	62.4	62.5	62.3

Table 8.1. B^3 results for ACE noun phrase coreference. SAMPLERANK-M is our proposed model that takes advantage of first-order features of the data and is trained with error-driven and rank-based methods. We see that both the first-order features and the training enhancements improve performance consistently.

x_i itself). Precision for x_i is defined as c_i divided by the number of noun phrases in x_i's cluster. Recall for x_i is defined as the c_i divided by the number of mentions in the gold standard cluster for x_i. $F1$ is the harmonic mean of recall and precision.

In addition to Pairwise Classification, Weighted Logic Classification, and Weighted Logic SampleRank-M, we also compare against Pairwise SampleRank-M, which differs from Weighted Logic SampleRank-M only by the fact that it is restricted to pairwise features.

Table 8.1 suggests both that first-order features and error-driven training can greatly improve performance. The Weighted Logic representation outperforms the Pairwise representation in F1 measure for both SampleRank training and classification training. We attribute some of this improvement to the capability of the Weighted Logic model to capture features of entire clusters that may indicate some phrases are not coreferent. Also, we attribute the gains from SampleRank training to the fact that inference is incorporated into the learning algorithm, as opposed to the pairwise classifier. This allows the learning algorithm to focus updates where they are needed by the inference algorithm.

It is interesting that even without the more complex features, **Pairwise SampleRank-M** outperforms **Pairwise Classification**. At first, this result seems to contradict the NER experiments in the previous section, where using a propositional representation with SampleRank did worse than using a propositional representation with CRF learning and inference. However, the difference here is that *even with the propositional representation,*

learning and inference are approximate. The improvement can therefore be attributed to the fact that SampleRank-M discovers more accurate parameters than the simple binary classification estimation.

Error analysis indicates that often noun x_i is correctly not merged with a cluster $\mathbf{x}^j$ when $\mathbf{x}^j$ has a strong internal coherence. For example, if all 5 mentions of *France* in a document are string identical, then the system will be extremely cautious of merging a noun that is not equivalent to *France* into $\mathbf{x}^j$, since this will turn off the "All-String-Match" feature for cluster $\mathbf{x}^j$.

To our knowledge, the best results on this dataset were obtained by the meta-classification scheme of Ng [79]. Although our train-test splits may differ slightly, the best B-Cubed F1 score reported in Ng [79] is 69.3%, which is considerably lower than the 79.3% obtained with our method. Also note that the Pairwise baseline obtains results similar to those in Ng and Cardie [80]. These results indicate that combining more expressive representations with suitable learning algorithms can improve over standard propositional approaches.

Finally, as a point of reference, inter-annotator agreement for the coreference task has been found to be between 80-90% [111].

8.4 Related Work

There have been a number of machine learning approaches to coreference resolution, traditionally factored into classification decisions over pairs of nouns [100, 80]. Denis and Baldridge [31] use a ranking loss function for pronoun coreference; however the examples are still pairs of pronouns, and the example generation is not error driven. Ng [79] learns a meta-classifier to choose the best prediction from the output of several coreference systems. While the meta-classifier can flexibly represent features, the method is neither error-driven nor rank-based.

Bhattacharya and Getoor [6] and Haghighi and Klein [47] both present unsupervised learning algorithms for coreference resolution that obtain competitive results despite not

using any labeled training data. However, it is not clear how to extend them to weighted logic representations.

McCallum and Wellner [67] use a conditional random field that factors into a product of pairwise decisions about pairs of nouns. These pairwise decisions are made collectively using relational inference; however, as pointed out in Milch et al. [72], this model has limited representational power since it does not capture features of *entities*, only of pairs of mentions. Milch et al. [73] address these issues by constructing a generative probabilistic model, where noun clusters are sampled from a generative process. Our current work has similar representational flexibility as Milch et al. [73] but is discriminatively trained.

CHAPTER 9

MULTI-TASK INFERENCE

9.1 Motivation

Weighted logic is particularly appealing for language processing because it provides a common formalism to piece together different components of a large system. For example, an end-to-end language system may include speech recognition, machine translation, word segmentation, part-of-speech tagging, phrase structure parsing, entity and relation extraction, and data mining. Current attempts to build such systems rely on bottom-up beam search; i.e., a set of hypotheses are passed from lower to higher levels of processing. However, there exist constraints in high level hypotheses that can greatly improve lower level processing. For example, all mentions of named-entities should be part of noun phrases. It is relatively straight-forward to write down these inter-component constraints; however, designing prediction algorithms that can process them efficiently has proven to be difficult.

We refer to the problem of performing inference in multiple related problems as *multi-task inference.* There are two common approaches to multi-task inference. The first approach solves the problems in a fixed order, passing a beam or set of weighted samples of solutions from one problem to the next [36, 60, 105]. The advantage of this approach is that they can be quite efficient since they use exact polynomial time algorithms to generate samples from each problem. The disadvantage is that information only flows in one direction, in the order the problems are solved.

The second type of approach disregards the decomposition into subtasks [89]. Instead, the set of problems is transformed into a single, large structured prediction problem, where the output space is the cross-product of the output spaces of each task. The advantage of

this approach is that by directly modeling dependencies between variables in different problems, both bottom-up and top-down dependencies can be captured. The disadvantage is that it replaces well-studied and successful *exact* solutions to subtasks with general-purpose approximate estimation and prediction algorithms such as Gibbs sampling or stochastic search.

In this section, we present two algorithms for multi-task inference that attempt to find a middle ground between these two approaches. The first is based on a synthesis of satisfiability solvers and belief propagation, in which one large weighted satisfiability problem is decomposed into a set of loosely coupled problems. A stochastic satisfiability solver is used to compute several likely assignments to each sub-task, then these potential assignments are transmitted across sub-tasks using message passing.

The second solution we propose a version of Metropolis-Hastings with a proposal distribution designed for multi-task inference. The resulting algorithm enables a weighted logic representation to capture long-range dependencies between tasks, and uses exact inference to sub-tasks as part of the proposal distribution to increase the efficiency of the sampler.

9.2 Sparse Generalized Max-Product

A common prediction algorithm for weighted logic is to first convert the input into a weighted maximum satisfiability problem, then apply an off-the-shelf solver to find a local maximum. A weighted maximum satisfiability problem (WMAX-SAT) has as input a Boolean formula in conjunctive normal form, where each clause has a corresponding weight. A solution is a truth assignment that maximizes the sum of the weights of the satisfied clauses.

Many important problems in artificial intelligence can be reduced to instances of WMAX-SAT. For example, Park [82] has shown that finding the most probable variable assignment in a Bayesian network can be solved efficiently through a reduction to WMAX-SAT. More

recently, Richardson and Domingos [93] have used WMAX-SAT solvers to find the most probable variable assignment in Markov Logic Networks.

Stochastic WMAX-SAT solvers such as Walksat [98] are efficient, scalable, and domain-independent. However, because of their generality, they ignore structural properties of the problem. For instance, many problems in artificial intelligence decompose into loosely-coupled subproblems, in which densely connected clusters of variables are connected to adjacent clusters by a relatively small number of clauses. In many cases, each of these subproblems can be solved with an efficient specialized algorithm (e.g., a dynamic program); the difficulty is in communicating solutions among subproblems. For example, language processing pipelines often consist of a set of specialized dynamic programs to perform tasks such as speech recognition and parsing. It would be unfortunate to replace these specialized, efficient algorithms with a generic, stochastic search method.

In this chapter, we propose a search algorithm for WMAX-SAT that is designed specifically for problems that exhibit this clustering of variables. Inspired by belief propagation algorithms that have been effective for probabilistic inference [117], our algorithm computes solutions to overlapping subproblems of the input, then iteratively passes messages between subproblems to converge upon a global solution. When the subproblems are solved using a stochastic solver such as Walksat, our algorithm can be understood as a way to augment stochastic search with belief propagation. The resulting algorithm provides a general framework for solving many related maximization problems jointly.

As we will show, the size of these messages grows quickly with the number of variables shared among clusters. We therefore introduce and compare several *sparse* representations of messages that allow us to compactly transmit summary information from one cluster to the next.

We empirically validate our approach on synthetic and real-world WMAX-SAT problems and demonstrate significant improvements in solution quality. We hypothesize two reasons for this improvement: First, by computing solutions to subproblems, we effectively

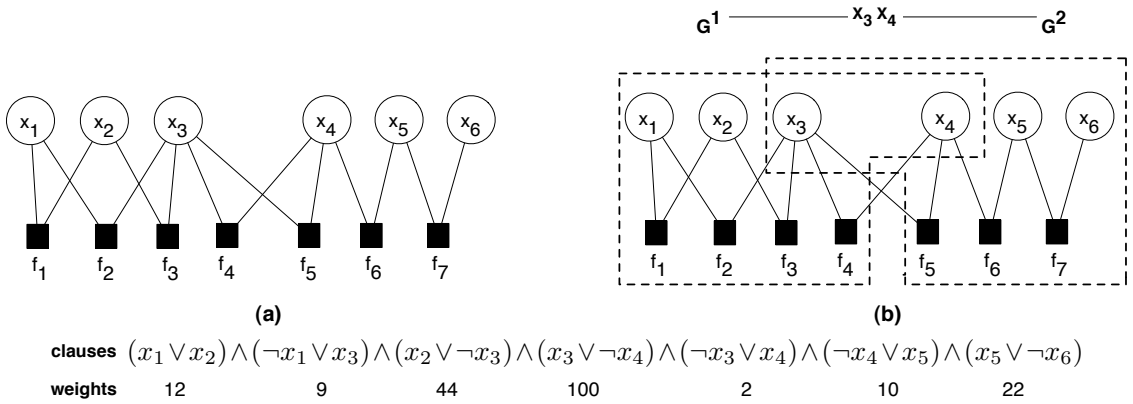

Figure 9.1. (a) A factor graph for a WMAX-SAT instance with 7 clauses and 6 variables. (b) A cluster graph for the same instance containing two clusters, G^1 and G^2.

"cache" partial solutions to the original problem. Thus, when the assignment to a variable in one subproblem is changed, we can quickly look-up the best compatible solution to a related subproblem. Second, the subproblems are often easier to solve than the original problem because they are less constrained (i.e. have a lower clause-to-variable ratio). We perform several experiments to support these claims.

9.2.1 Weighted Maximum Satisfiability

In this section, we first formally define WMAX-SAT then describe solutions based on max-product and generalized max-product algorithms. We then propose an algorithm we call *sparse generalized max-product*, present several experiments validating our approach, discuss related work, then conclude with a list of proposed research.

Let $\mathbf{X}$ be a set of n boolean variables $\{X_1 \dots X_n\}$, and let $\mathbf{c}$ be a set of m *clause functions* $\{c_1 \dots c_m\}$, where $c_i(\mathbf{X}_s \subseteq \mathbf{X})$ maps a set of variables to a real number. Let $\delta(\mathbf{X}_s) \in \{0, 1\}$ be an indicator function that is 1 if and only if there exists $X_i \in \mathbf{X}_s$ such that $X_i = 1$; that is, $\delta(\mathbf{X}_s)$ indicates if the disjunction of the variables in $\mathbf{X}_s$ evaluates to true. Clause $c_i(\mathbf{X}_s)$ is said to be *satisfied* if $\delta(\mathbf{X}_s) = 1$. The value of a clause is defined as $c_i(\mathbf{X}_s) = w_i\delta(\mathbf{X}_s)$, where w_i is the *weight* for clause c_i.

Let $\mathbf{x} \in \{0,1\}^n$ be an assignment to $\mathbf{X}$. The *score* of an assignment is the sum of the weights of the satisfied clauses: $s(\mathbf{X} = \mathbf{x}) = \sum_{c_i \in \mathbf{c}} c_i(\mathbf{X}_s = \mathbf{x}_s)$. *Weighted maximum satisfiability (WMAX-SAT)* is the problem of finding the assignment with the highest score:

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} s(\mathbf{X} = \mathbf{x}) \tag{9.1}$$

Many efficient algorithms have been proposed to solve WMAX-SAT. (Stützle et al. [102] provide an overview.) Most are local search algorithms that avoid local minima either by injecting randomization (e.g., Walksat [54]) or by exploiting search history (e.g., Tabu Search [44]). Given the efficiency and generality of these techniques, it is reasonable to encode many difficult optimization problems as WMAX-SAT instances and solve them with an off-the-shelf solver. Indeed, this approach has been taken in probabilistic reasoning to find the most likely assignment to variables in a graphical model in cases where the exact dynamic programming solution is intractable [82, 93].

However, this "reduce to SAT" approach ignores structural properties of the original problem. In this chapter, we consider an important class of problems that can be viewed as a combination of subproblems. Each subproblem is represented by a set of clauses over a subset of $\mathbf{X}$, and two subproblems interact via the variables they share. We refer to this class of problems as *clustered* WMAX-SAT instances.

Solving clustered WMAX-SAT instances with a standard solver fails to take advantage of this structure. This is especially problematic if there exist efficient, exact solutions to subproblems, as these exact solutions will only be approximated with a general WMAX-SAT solver. For example, if parsing is one subproblem of a larger NLP task, it would be preferable to use the exact dynamic programming solution to perform parsing, rather than approximate search. In the following section, we describe message passing algorithms for WMAX-SAT, and then propose an algorithm that is particularly suited to clustered WMAX-SAT problems.

9.2.2 A Graphical Model for WMAX-SAT

We will use the language of graphical models to represent WMAX-SAT problems. For a review of graphical modeling terminology, see Section 2.1. Given a factor graph representing distribution $p(\mathbf{X}) \propto \prod_i f_i(\mathbf{X}_s)$, a common inference task is to find the most probable assignment to $\mathbf{X}$:

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} p(\mathbf{X} = \mathbf{x}) \tag{9.2}$$

$$= \operatorname*{argmax}_{\mathbf{x}} \sum_i \log f_i(\mathbf{X}_s = \mathbf{x}_s) \tag{9.3}$$

Substituting $\log f_i(\mathbf{X}_s = \mathbf{x}_s) = c_i(\mathbf{X}_s = \mathbf{x}_s)$ in Equation 9.3 results in an optimization problem equivalent to the WMAX-SAT problem given in Equation 9.1, where factors represent clauses, and the probability of an assignment is proportional to its score.

A distribution can be represented by a *factor graph* $\mathcal{G} = (\mathbf{X}, \mathbf{f}, \mathbf{E})$, a bipartite graph where vertex set X is a set of random variables, f is the set of factors, and edges E connect factors to their arguments (see Figure 9.1(a)). If $\mathcal{G}$ is acyclic, then the well-known *max-product* message passing algorithm can exactly compute the most probable assignment to $\mathbf{X}$ in polynomial time [114]. When $\mathcal{G}$ contains cycles, max-product provides an approximate solution; however, not only is there no guarantee that this approximation will return the optimal solution, but there is also no guarantee that the algorithm will converge. There are two additional difficulties of using max-product to solve WMAX-SAT. First, it is non-trivial to convert the output of max-product into an optimal assignment to $\mathbf{X}$ because max-product provides a distribution over assignments to single variables, not a joint distribution over all variables.

A common but expensive workaround is the iterative *decimation* procedure described in Braunstein et al. [10], in which variables are incrementally set to their most likely assignments and $\mathcal{G}$ is simplified between calls to max-product. Second, much like approximate solvers for WMAX-SAT, max-product ignores any structure in the connectivity of the variables. Thus, max-product is not well-suited to clustered WMAX-SAT instances.

9.2.3 Sparse Generalized Max-Product for WMAX-SAT

Yedidia et al. [117] introduced *generalized belief propagation* to perform inference in graphical models that contain clusters of variables. We refer to the maximization version of generalized belief propagation as *generalized max-product*. In this section, we first describe how to apply generalized max-product to WMAX-SAT and discuss the difficulties involved. We then propose ways to overcome these difficulties by extending generalized max-product to operate with sparse messages and stochastic WMAX-SAT solvers.

Let the original factor graph be $\mathcal{G} = (\mathbf{X}, \mathbf{E}, \mathbf{f})$. Partition the factors of $\mathcal{G}$ into t disjoint subsets $\{f_1 \dots f_t\}$. Define the *cluster graph* $\mathcal{G}_c$ to have t overlapping subgraphs $\{G^1 \dots G^t\}$ called *clusters*, where $G^k = (\mathbf{X}^k, \mathbf{E}^k, \mathbf{f}^k)$, $\mathbf{X}^k \subseteq \mathbf{X}$ and $\mathbf{f}^k \subseteq \mathbf{f}$ (Figure 9.1(b)). Note that each factor appears in exactly one cluster, but variables may appear in more than one cluster. For example, each cluster may correspond to a subproblem of the original input.

$\mathbf{X}^k$ contains all variables that are arguments to factors in $\mathbf{f}^k$. Let $\mathbf{X}^{ij}$ be the variables shared between cluster G^i and G^j. $\mathcal{G}_c$ contains an edge between two clusters G^i and G^j if they share variables, i.e., $\mathbf{X}^{ij} \neq \emptyset$. Each edge in $\mathcal{G}_c$ is labeled with $\mathbf{X}^{ij}$.

The product of the factors in each of the subgraphs is equal to the product of the factors in the original graph, that is $\prod_{f_i \in \mathbf{f}} f_i(\mathbf{x}_s) = \prod_{G^i \in \mathcal{G}_c} \prod_{f_i \in \mathbf{f}^i} f_i(\mathbf{x}_s)$. We define $s^i(\mathbf{x}^i) = \prod_{f_i \in \mathbf{f}^i} f_i(\mathbf{x}_s)$ as the local score of G^i.

Generalized max-product can be understood as the max-product algorithm applied to $\mathcal{G}_c$. Rather than computing distributions over single variables, the algorithm returns a distribution over the joint assignment to variables in each cluster. When clusters correspond to subproblems of the original input, this can be understood as computing a distribution over solutions to each subproblem, then iteratively modifying the solutions of each subproblem to reflect their viability as part of a global solution. The message transmitted from cluster i to cluster j is

$$m_{i \to j}(\mathbf{X}^{ij} = \mathbf{x}^{ij}) = \max_{\mathbf{x}^i : \mathbf{X}^{ij} = \mathbf{x}^{ij}} s^i(\mathbf{x}^i) \prod_{k \neq j} m_{k \to i}(\mathbf{x}^{ik}) \tag{9.4}$$

When $\mathcal{G}_c$ contains cycles, these messages are iteratively computed until convergence, at which point the *belief* of cluster G^i is defined as

$$b_i(\mathbf{X}^i = \mathbf{x}^i) = s^i(\mathbf{x}^i) \prod_j m_{j \to i}(\mathbf{x}^{ij}) \tag{9.5}$$

Thus, $m_{i \to j}(\mathbf{X}^{ij})$ is a vector representing a distribution over scores for assignments of $\mathbf{X}^{ij}$, and $b_i(\mathbf{X}^i)$ is a vector representing a distribution over scores for $\mathbf{X}^i$.

When $\mathcal{G}_c$ is acyclic, the resulting beliefs are exact; otherwise, generalized max-product offers no better theoretical guarantees than traditional max-product. However, generalized belief propagation has been shown to often have better empirical convergence rates than traditional belief propagation [117]. More importantly, generalized max-product is appealing for clustered WMAX-SAT instances because it directly accounts for problem structure. It also allows us to substitute specialized inference algorithms (when they exist) to efficiently compute solutions to subproblems.

There are two primary difficulties with applying generalized max-product to WMAX-SAT. First, the size of message $m_{i \to j}(\mathbf{X}^{ij})$ grows exponentially with $|\mathbf{X}^{ij}|$. Second, the maximization over $\mathbf{x}^i$ in Equation 9.4 naively requires an exponential search over assignments to $\mathbf{X}^i$. As it is not uncommon for $|\mathbf{X}^i|$ to be on the order of thousands of variables, we must consider alternatives to brute-force enumeration. The following sections address these issues in turn.

9.2.3.1 Representing sparse beliefs and messages

Ordinarily, $m_{i \to j}(\mathbf{X}^{ij})$ and $b_i(\mathbf{X}^i)$ are represented by a table with one entry per complete assignment to $\mathbf{X}^{ij}$ or $\mathbf{X}^i$. We refer to these as *complete messages* and *complete beliefs*.

When the complete messages grow too large, a simple approximation is to store a fully-factored message containing one score distribution per variable. Each table entry is the score of the best complete assignment containing the specified variable assignment. We

refer to these as *univariate messages* and *univariate beliefs*. This is similar to the messages advocated recently in Duchi et al. [33].

While univariate messages are compact, they do not represent interactions among variables, especially when they must be approximated with message passing. Furthermore, if there are ties in the univariate messages, selecting the maximum variable setting for each independently is not guaranteed to return the optimal joint assignment [84].

A compromise between these two extremes is a particle-based message. In this case, a particle is simply a row of the complete table, and a message is approximated by a collection of particles. A simple special case of a particle message is an n-best list. An n-best list stores scores for joint assignments as in complete messages, but the low scores are simply truncated from the table. This representation allows the messages to be sensitive to variable interactions while limiting memory requirements. We refer to these as *n-best messages* and *n-best beliefs*.

9.2.3.2 Computing sparse beliefs and messages

Normally, the maximization in Equation 9.4 is computed by iterating over assignments to $\mathbf{X}^i$ and multiplying the assignment's local score with the score for its corresponding incoming message. Because the number of assignments is too large to iterate for each message, we must investigate an alternative way to compute the maximization.

In this section we present an algorithm to pass sparse messages within generalized max-product. The first step is to initialize the sparse belief $b_i(\mathbf{X}^i)$ for each cluster. If a cluster has a specialized maximization algorithm such as a dynamic program, we can modify this algorithm to return the desired sparse representation. For example, to compute an n-best belief, we can simply compute the top n solutions using the maximization algorithm. If no specialized algorithm exists, we can treat this computation as a generic WMAX-SAT problem and use any off-the-shelf solver. In our experiments, we use Walksat [54], a highly-

scalable stochastic solver. We refer to the method that computes these initial beliefs as *ComputeUnconstrainedBeliefs*.

Given this initial assignment, the next step is to update each cluster's sparse beliefs given the sparse beliefs of its neighboring clusters. Let $\mathbf{X}^i \setminus \mathbf{X}^{ik}$ be the set of variables contained in cluster G^i but not in G^k. As shown in Equation 9.4, to compute message $m_{i \to j}(\mathbf{X}^{ij})$, we must find the maximum setting of $\mathbf{X}^i \setminus \mathbf{X}^{ik}$ *given* each assignment to $\mathbf{X}^{ik}$ stored in the incoming message from G^k.

We view this as a set of *constrained maximization problems*. For each incoming assignment to $\mathbf{X}^{ik}$, we must find the best setting(s) of $\mathbf{X}^i \setminus \mathbf{X}^{ik}$ *constrained* such that $\mathbf{X}^{ik} = \mathbf{x}^{ik}$. The local score of this best assignment is combined with the score of the incoming message to update $b_i(\mathbf{X}^i)$.

If a cluster has a specialized maximization algorithm, we can modify it to return constrained solutions, e.g. by constrained dynamic programming. Otherwise, we can treat this constrained maximization problem as a WMAX-SAT instance. Given an assignment $\mathbf{x}^{ik}$, we can simplify the factors in G^i (i.e. by removing clauses in G^i satisfied by $\mathbf{x}^{ik}$ and removing variables in $\mathbf{X}^{ik}$ from unsatisfied clauses). We can now again use any off-the-shelf solver to compute constrained solutions (e.g. Walksat). We refer to the method that solves this constrained maximization problem as *ComputeConstrainedBelief*.

There are two additional details about *ComputeConstrainedBelief*. First, very often we can simply lookup the best assignment to $\mathbf{X}^i \setminus \mathbf{X}^{ik}$ given $\mathbf{x}^{ik}$ because a compatible solution may already exist in $b_i(\mathbf{X}^i)$. Second, because each round of message passing may increase the size of $b_i(\mathbf{X}^i)$, we may need to re-compress the belief, for example, by storing only the n-best assignments or recomputing the univariate values.

Message passing proceeds until either a specified number of iterations is reached or the difference in beliefs between iterations falls below a threshold. If the top beliefs of each cluster are incompatible at the end of message passing, we can use a decimation procedure similar to that used in Braunstein et al. [10], although here we can decimate using

Algorithm 5 Sparse Generalized Max-Product

Input: $\mathcal{G}_c$
// Initialize beliefs
for all $G^i \in \mathcal{G}_c$ **do**
  $b_i(\mathbf{X}^i) \Leftarrow$ *ComputeUnconstrainedBelief*(G^i)
end for
// Perform message passing
while Not Converged **do**
  // for each cluster
  for all $G^i \in \mathcal{G}_c$ **do**
    // for each neighboring cluster
    for all G^j s.t. $\mathbf{X}^{ij} \neq \emptyset$ **do**
      // for each incoming assignment
      for all $\mathbf{x}^{ij}$ s.t. $m_{j \to i}(\mathbf{X}^{ij} = \mathbf{x}^{ij}) \neq 0$ **do**
        ComputeConstrainedBelief$(G^i, \mathbf{x}^{ij})$
      end for
    end for
  end for
end while

clusters, rather than individual variables. Pseudo-code for the final algorithm is presented in Algorithm 5.

9.2.4 Maximum Satisfiability Experiments

The goal of these experiments is to understand under what conditions augmenting an off-the-shelf WMAX-SAT solver with a message passing protocol can improve performance. We vary both the characteristics of the problems as well as the type of message representation. In all experiments, we use the weighted version of Walksat [54], which we ported to Java from the original C implementation[1]. Note that Walksat is used both for comparison and as a subroutine to compute messages.

For the first set of experiments, we randomly generate WMAX-SAT instances that exhibit a clustering of variables. We implement a variant of the *Cluster3-SAT* problem generator given in Frank et al. [37]. We first generate k 3-SAT subproblems uniformly at random,

[1] `www.cs.rochester.edu/u/kautz/walksat/`

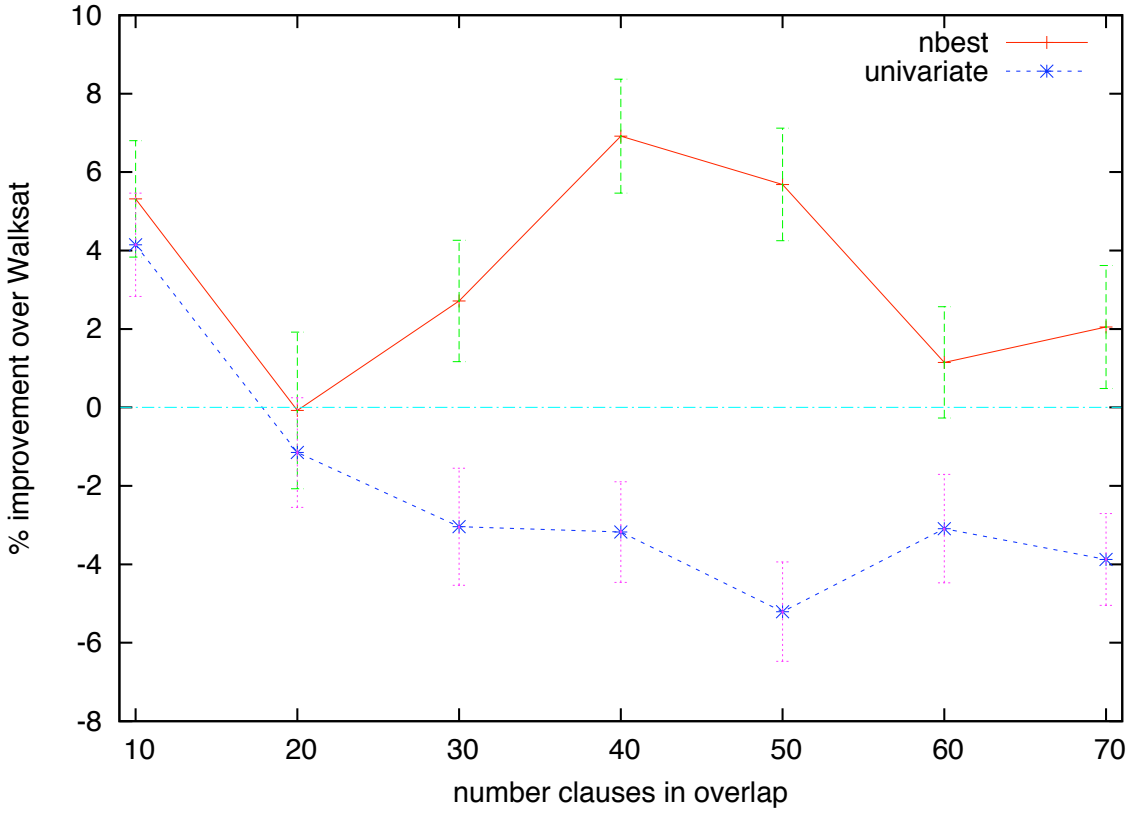

Figure 9.2. Comparison of n-best and marginal messages as the number of clauses containing shared variables increases.

then randomly link the problems together by generating clauses that contain variables from each subproblem. Weights for each clause are sampled from a Gaussian distribution. For these experiments, we set the number of particles per message $n = 50$.

To compare our message passing algorithms with Walksat, we first run the message passing algorithm and record the score of the best assignment and the total number of Walksat iterations it required. We then run traditional Walksat on the original problem for the same number of iterations and record the score of the best assignment. This allows us to measure the efficiency of each search algorithm. For each setting of the problem generator, we generate 100 samples and average the results. In the following experiments, we compare performance across different message types, and across different numbers of variables, clauses, and clusters.

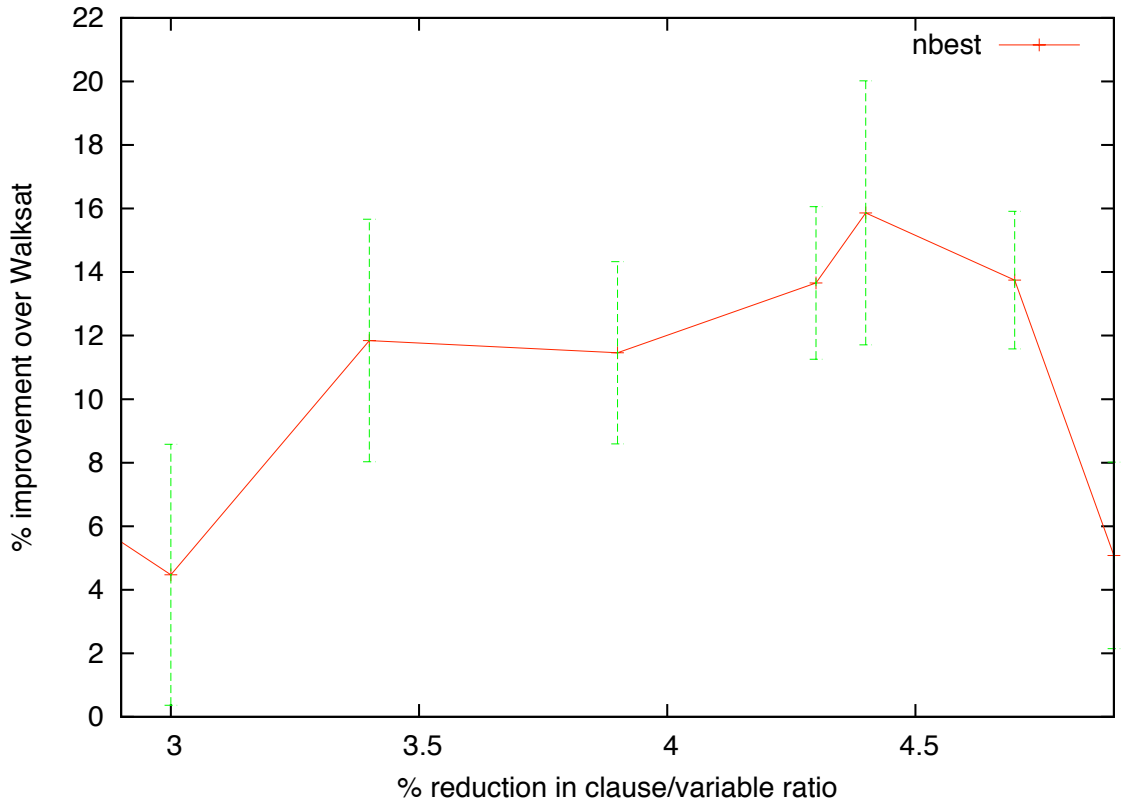

Figure 9.3. Correlation between reduction in clause-variable ratio and score improvement for n-best messages.

Figure 9.2 compares univariate and n-best messages on 700 instances with 100 variables, 400 clauses, and 2 clusters. The x-axis is the number of clauses containing variables that are shared between the two clusters. This figure supports our intuition that as the overlapping variables become more constrained, representing their distribution with a univariate message fails to capture the compatibilities among variables. We do not have a full explanation for the drop in performance for 20 clauses, although it may be related to the phase transition phenomenon that has been observed in unweighted satisfiability problems as the relative number of constraints increases.

In Figure 9.3, we test the hypothesis that message passing outperforms Walksat because of a reduction in the clause-variable ratio. Recall that the cluster graph is constructed such that two clusters may share variables but not factors. Thus, the average clause-variable

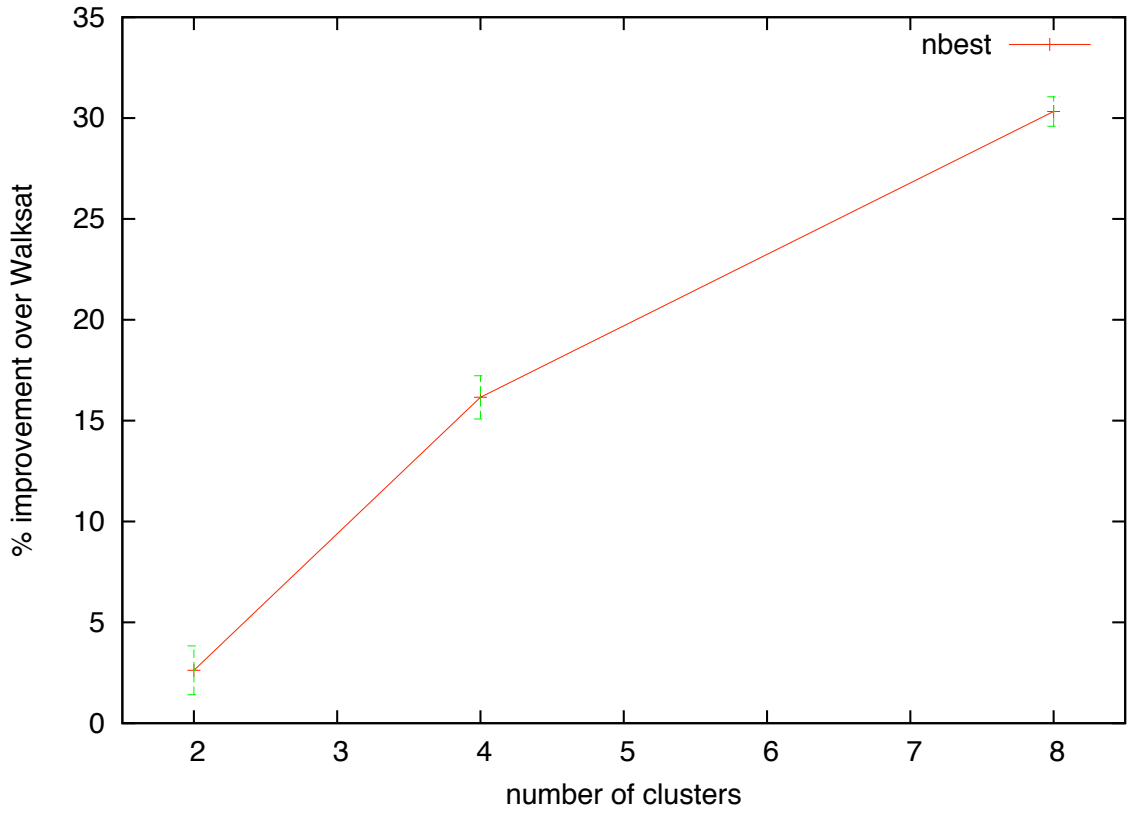

Figure 9.4. Comparison of improvement of n-best over Walksat as the number of clusters increases.

ratio per cluster is less than that of the original problem. Figure 9.3 plots the percent improvement of n-best message passing over Walksat as the reduction in the clause-variable ratio increases. These experiments sample 175,000 different instances containing 80 to 180 variables, 300 to 600 clauses, and 2 clusters. The results provide correlational evidence in support of our hypothesis; however, the results have considerable variance that makes the results somewhat inconclusive.

Figure 9.4 plots performance as the number of clusters increases. We generate 300 instances with 400 variables, 1600 clauses, and the number of clusters varying from 2 to 8. In each case, the clusters are fully connected, with each pair of clusters sharing two variables. Figure 9.4 shows a significant increase in performance as the divisibility of the problem increases. That is, the greater number of sub-tasks the original problem can be

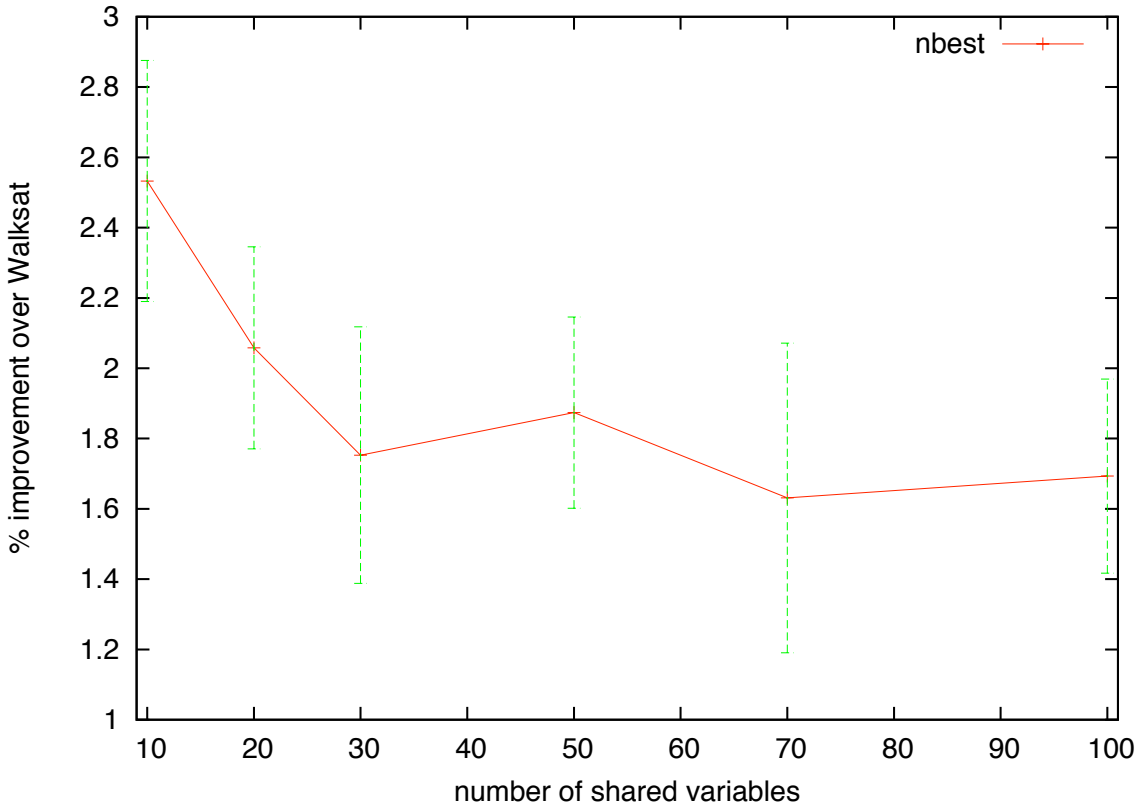

Figure 9.5. Comparison of improvement of n-best over Walksat as the number of variables shared between clusters increases.

divided into, the greater the gains obtained from clustering the graphical model. Note that this improvement is amplified by the relatively small number of shared variables, which reduces the chances of a good solution being excluded by message compression.

Figure 9.5 plots performance as the number of variables shared between clusters increases. We generate 700 instances with 1000 variables, 4000 clauses, and 2 regions, and vary the number of shared variables from 5 to 100. This figure indicates that performance is best with few shared variables, but message passing still outperforms Walksat even as the potential message size increases to 2^{100}.

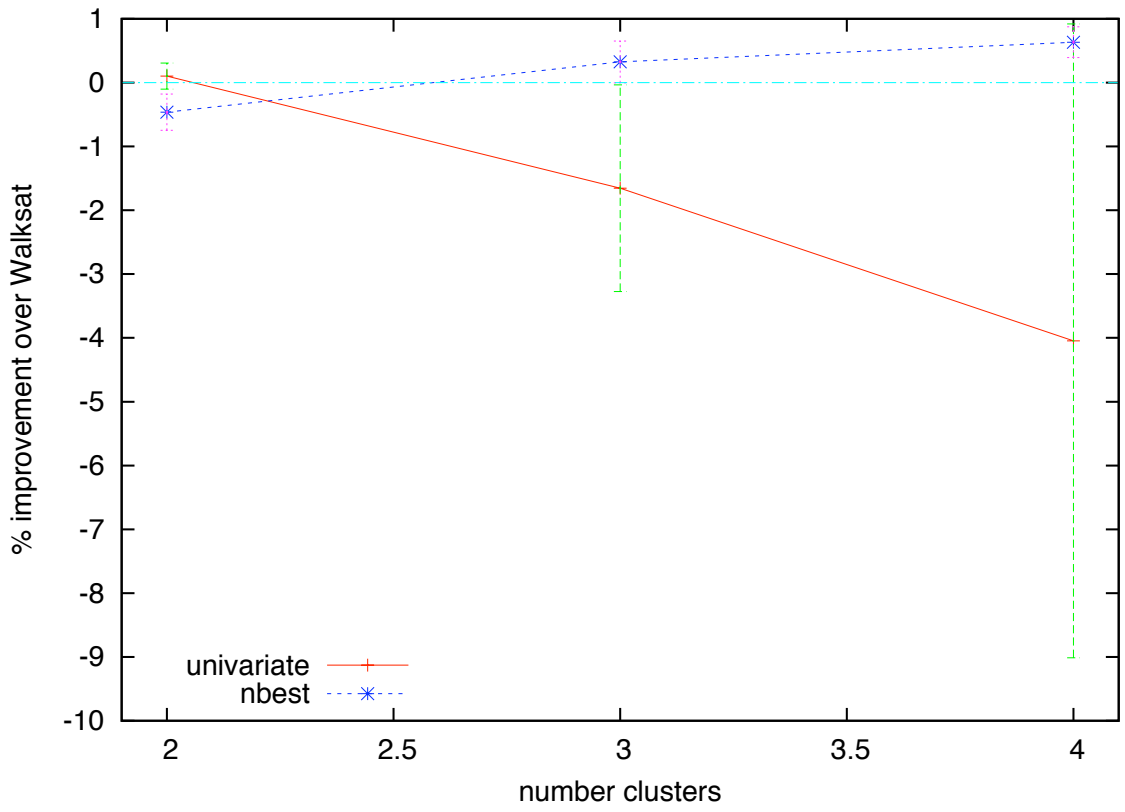

Figure 9.6. Comparison of Walksat and message passing algorithms on the *SPOT5* data.

Finally, we test our algorithm on a real-world WMAX-SAT instance that is part of the benchmarks used in the MaxSAT-2006 competition.[2] In particular, we sample a problem from the *SPOT5* corpus, which was generated from a satellite scheduling domain [3]. To partition the problem into clusters, we use a well-known graph-theoretic technique based on betweenness-centrality [113].

Figure 9.6 shows results averaged over 100 trials for an instance with 129 variables and 1037 clauses. We set $n = 100$. We can see that univariate message passing improves slightly over Walksat when the problem is split into 2 clusters, but performs worse with larger clusters. On the other hand, n-best message passing outperforms Walksat when

[2] `www.iiia.csic.es/~maxsat06/ms06-bench.tgz`

there are 3 and 4 clusters. This is appealing, because an examination of the factor graph for this problem reveals 4 well-defined clusters.

In conclusion, we have presented a new message passing algorithm to solve maximization problems that consist of multiple overlapping subproblems. We have empirically demonstrated the validity of this approach on synthetic and real-world data, and we have evaluated hypotheses to explain its performance.

9.3 Multi-Task Metropolis-Hastings

While the algorithm in the previous section has shown potential to perform well on multi-task inference problems, its reliance on Walksat as an underlying search algorithm presents the problem of "full propositionalization" methods, as discussed in Chapter 3. Namely, problems with a large number of deterministic constraints would need to explicitly instantiate all of these factors before using Walksat. For many problems, this is impractical.

In this section, we return to the sampling-based methods proposed in Chapter 3. We augment standard Metropolis-Hastings sampling with a proposal distribution designed for multi-task inference. More specifically, we use a weighted logic representation to capture long-range dependencies between tasks, but we preserve the exact estimation and prediction algorithms for the subtasks. Our approach samples from this complex, global model by using the submodels as a proposal distribution in Metropolis-Hastings.

9.3.1 Previous Work

We begin by defining a multi-task inference problem as a set of inference problems $\mathbf{S} = \{S^1 = \langle \mathbf{x}^1, \mathbf{y}^1 \rangle \dots S^n = \langle \mathbf{x}^n, \mathbf{y}^n \rangle\}$. Let $\mathbf{y} = \{\mathbf{y}^1 \dots \mathbf{y}^n\}$ be an assignment to the set of all output variables, and let $\mathbf{x} = \{\mathbf{x}^1 \dots \mathbf{x}^n\}$ be an assignment to the set of all input variables.

Perhaps the simplest solution to multi-task inference is beam search. Beam search operates by first fixing the order in which tasks will be solved. A set of weighted solutions

to the first task is computed and passed downstream to the subsequent task. The set of solutions for the next task are then computed *conditioned* on each element of the solution set from the earlier tasks. The final solution chosen is the one with the highest combined score across all tasks.

Finkel et al. [36] present an enhancement to beam search that performs forward-sampling on the *directed graphical model* (i.e., Bayes net) induced by the task ordering. Let $\mathbf{pa}(\mathbf{y}^i)$ be the *parent set* of task S^i, where $\mathbf{y}^j \in \mathbf{pa}(\mathbf{y}^i)$ if the solution of task S^j is required to compute the solution to S^i. Given a set of probability distributions $p(\mathbf{y}^i|\mathbf{x}^i, \mathbf{pa}(\mathbf{y}^i))$, a conditional distribution over all output variables is defined as follows:

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} p(\mathbf{y}^i|\mathbf{x}^i, \mathbf{pa}(\mathbf{y}^i)) \tag{9.6}$$

Because the conditional probability tables needed to compute Equation 9.6 grow exponentially with the output space of each task, approximate inference methods must be used. In Finkel et al. [36], the authors propose repeatedly drawing samples according to the ordering of the graph and selecting the best overall solution by majority voting.

While conceptually appealing and not difficult to implement given exact algorithms for each subtask, this approach suffers from the traditional limitations of directed graphical models. Namely, it assumes that the task graph contain no cyclic dependencies. This means that the solution to a parent node cannot be conditioned on the solution to a child node. Furthermore, in the forward-sampling approach, samples flow in one direction only, according to the sampling order. The practical implications of these drawbacks is that errors made on one subtask cannot be corrected by downstream predictions.

In general, we would like to model arbitrary dependencies among output variables, regardless of the tasks to which they belong and the order in which tasks are solved. In the next section, we propose an undirected graphical model for multitask structured prediction that has these capabilities.

9.3.2 An Undirected Model for Multi-Task Inference

Given the set of input variables for all tasks $\mathbf{x} = \{\mathbf{x}^1 \dots \mathbf{x}^n\}$, we want to be able to model an arbitrary set of dependencies among the output variables of different tasks. To do this, we can use weighted logic to specify arbitrary formulae over the outputs of multiple tasks. From this weighted logic representation, we can construct a grounded CRF to model the conditional distribution over the set of output variables for all tasks $\mathbf{y} = \{\mathbf{y}^1 \dots \mathbf{y}^n\}$:

$$p(\mathbf{y}|\mathbf{x}; \Lambda) = \frac{1}{Z(\mathbf{x})} \prod_i f_i(\mathbf{y}, \mathbf{x}, \Lambda) \tag{9.7}$$

Since each factor can refer to variables from different tasks, this general formulation allows one to construct arbitrary feature functions that measure the compatibility of predictions from any set of subtasks. The factors can interact in complex ways because they are globally normalized by $Z(\mathbf{x})$, as opposed to the directed model, in which the factors in are normalized locally in each task. However, this flexibility comes at a cost: in general, we cannot guarantee the existence of polynomial-time exact inference algorithms. In particular, computing $Z(\mathbf{x})$ is intractable because it requires summing over an exponential number of assignments to $\mathbf{y}$, and the choice of factors will likely preclude dynamic programming solutions.

As described thus far, this model is an example of the second type of approach to multi-task inference described in the Section 9.1 — it flexibly captures bottom-up and top-down dependencies, but ignores exact solutions to subtasks. However, in the next section, we present an approximate inference algorithm for this model that leverages exact solutions for subtasks to perform sampling.

9.3.3 A Multi-Task Proposal Distribution

Recall from Section 4.2.2 that the Metropolis-Hastings algorithm generates a sample from a *target distribution* $p(\mathbf{y})$ by first sampling a new assignment $\mathbf{y}'$ from a simpler dis-

tribution $q(\mathbf{y}'|\mathbf{y})$, called the *proposal distribution*. The new assignment $\mathbf{y}'$ is retained with probability $\alpha(\mathbf{y}'|\mathbf{y}) = \min\left(1, \frac{p(\mathbf{y}')q(\mathbf{y}|\mathbf{y}')}{p(\mathbf{y})q(\mathbf{y}'|\mathbf{y})}\right)$. Otherwise, the original assignment $\mathbf{y}$ is kept.

The proposal distribution $q(\mathbf{y}'|\mathbf{y})$ can be a nearly arbitrary distribution, as long as it can be sampled from efficiently and satisfies certain constraints to ensure that the stationary distribution of the Markov chain is indeed $p(\mathbf{y})$.

The core of our approach in this section is to sample from the target distribution in Equation 9.7 using a proposal distribution composed of the exact polynomial-time inference algorithms to the subtasks.

In addition to the globally normalized target distribution $p(\mathbf{y}|\mathbf{x};\Lambda)$, assume we are given a set of local conditional models $\{p(\mathbf{y}^1|\mathbf{x}^1, \mathbf{n}(\mathbf{y}^1)) \dots p(\mathbf{y}^n|\mathbf{x}^n, \mathbf{n}(\mathbf{y}^n))\}$, where $\mathbf{n}(\mathbf{y}^i)$ are the *neighbors* of task S^i. That is, $\mathbf{y}^j \in \mathbf{n}(\mathbf{y}^i)$ if the solution to S^i depends upon the solution to S^j. (Note that unlike in the directed model described earlier, we do not assume acyclicity in the neighborhood graph.)

From these local models, we will construct a proposal distribution. An efficient proposal distribution is one that generates samples that are likely to be accepted by the target distribution. For example, generating assignments at random according to the local models is likely to be inefficient — not only might most of the proposals be rejected, but the ones that are accepted may be quickly forgotten by the next proposal.

Instead, we construct a proposal distribution that combines a Gibbs sampling step for one task, then propagates the effects of this sample to the remaining tasks. The pseudo-code is presented in Algorithm 6.

To generate a new sample $\mathbf{y}^t$, first a task i is selected uniformly at random (line 3). Next, a particular variable y^i_j in task i is selected to be changed (line 4). As in Gibbs sampling, this selection could be made at random, or according to a fixed schedule. A possibly new assignment is generated for variable y^i_j according to the conditional distribution provided by the global model (line 5). As in Gibbs sampling, we can construct this conditional distribution by normalizing over all possible assignments to y^i_j

Algorithm 6 Multi-task Metropolis-Hastings Sampler

1: **Input:**
   global model $p(\mathbf{y}|\mathbf{x};\Lambda)$
   local models $p(\mathbf{y}^1|\mathbf{x}^i,\mathbf{n}(\mathbf{y}^1))\ldots p(\mathbf{y}^n|\mathbf{x}^i,\mathbf{n}(\mathbf{y}^n))$
   initial assignment $\mathbf{y}^0$
2: **for** $t \leftarrow 0$ to NumberSamples **do**
3: Sample starting task to sample $i \in \{1\ldots n\}$ u.a.r.
4: Select j, the variable in task i to be sampled.
5: Sample assignment to $y^i_j \sim p(y^i_j|\mathbf{y}\setminus\mathbf{y}^i_j,\mathbf{x})$
6: **for all** $k \in \{1\ldots n\}$, $k \neq i$ **do**
7: Sample assignment $\mathbf{y}^k \sim p(\mathbf{y}^k|\mathbf{x}^k,\mathbf{n}(\mathbf{y}^k))$
8: **end for**
9: Set $\mathbf{y}'$ to be union of sampled assignments for each task.
10: $\alpha = \min\{1, \frac{p(\mathbf{y}')q(\mathbf{y}^t|\mathbf{y}')}{p(\mathbf{y}^t)q(\mathbf{y}'|\mathbf{y}^t)}\}$
11: with probability α: $\mathbf{y}^{t+1} \Leftarrow \mathbf{y}'$
12: with probability $(1-\alpha)$: $\mathbf{y}^{t+1} \Leftarrow \mathbf{y}^t$
13: **end for**

$$p(y^i_j|\mathbf{y}\setminus y^i_j,\mathbf{x}) = \frac{\prod_k f_k(\mathbf{y}\setminus y^i_j, y^i_j,\mathbf{x})}{\sum_{y^i_j}\prod_k f_k(\mathbf{y}\setminus y^i_j, y^i_k,\mathbf{x})}$$

Given this sampled assignment to y^i_j, the next step is to loop over all other tasks and resample according to the local model for each task. In the cases where the local model admits exact inference, we can exactly sample from this distribution.

Finally, a complete sample is constructed by concatenating all the assignments for each sub-task. This final sample is accepted or rejected according to the standard Metropolis-Hastings acceptance probability given in line 10.

The value of the proposal distribution $q(\mathbf{y}|\mathbf{y}')$ is the probability of the new assignment according to the product of each of the local models, i.e.:

$$q(\mathbf{y}|\mathbf{y}') = \prod_i p(\mathbf{y}^i|\mathbf{x}^i,\mathbf{n}(\mathbf{y}^i))$$

Note that this proposal distribution is actually independent of the previous assignment $\mathbf{y}'$. This can be problematic, because the sampler can "forget" many of the good samples accepted by the target distribution. To address this issue, in the experiments below we restrict

the sample of $\mathbf{y}^k$ in line 7 to only allow changes to a fixed number of variables from task k. The choice of which variables are allowed to change is problem dependent. In our experiments, each subtask is a linear-chain sequence model operating over the same input sequence. So, if variable y_j^i is sampled in task i, for a neighboring task k we only allow changes to variables $y_{j-1}^k, y_j^k, y_{j+1}^k$. That is, we allow changes to the three variables with the highest dependence on the value of y_j^i.

9.3.4 Joint Part-of-speech and Named Entity Recognition Experiments

To evaluate Multi-Task Metropolis-Hastings, we perform experiments on the combination of two problems: part-of-speech tagging (POS) and named entity recognition (NER). POS labels are often valuable input features to NER. Typically, POS tags are predicted by one system, then these labels are used as features by the NER system. In this section, we design a joint model that performs learning and inference over both tasks simultaneously.

9.3.4.1 Data

We use a subset of the CMU Seminars dataset[3], a collection of academic seminar announcements from Carnegie Mellon University. The types of entities are start time, end time, location, and speaker. We use a version of the dataset that has been additionally labeled with POS labels using the Brill tagger [11]. While these POS labels are not annotated by humans, the Brill tagger has been trained on much more labeled data, and is quite accurate (around 96%). We sample 12,075 tokens (791 sentences) from this dataset and perform 3-fold cross validation. Additionally, we collapse some of the part of speech tags (since very fine-grained tags are unlikely to be relevant to NER) resulting in 19 total POS tags.

For both tasks, we use the following features:

- The token itself.

[3] `http://www.cs.umass.edu/~mccallum/data/sa-tagged.tar.gz`

- Regular expressions indicating whether the token is punctuated, contains digits, contains capitalization, contains only capitalization, or begins with capitalization.
- The above features for positions before and after the current token.

In addition, the joint model uses the predicted POS tag (also offset before and after each token) as a feature for NER.

For the weighted logic models, we also include the following non-local features:

- The sentence contains no NER labels (i.e., only background labels).
- The sentence contains more than one mention of an entity type (e.g., "contains more than one end time").
- The sentence has no part-of-speech tag for one of {noun, verb, punctuation}.
- Label X occurs before label Y (e.g., "end time occurs before begin time").

9.3.4.2 Systems

We evaluate the following systems:

- **BeamSearch**: One linear-chain CRF is trained on POS independently. Another CRF is trained on NER given the "true" POS labels (as predicted by the Brill tagger). For a testing sentence, the top 50 most probable POS solutions are generated from the first CRF. For each, the most probable NER solution is calculated according to the second CRF. The assignment that maximizes the product of the POS and NER assignment probabilities is chosen.
- **SampleRank**: The SampleRank learning algorithm using the MIRA++ update. The configuration uses Gibbs sampling with temperature 0 (i.e., coordinate-ascent), best neighbor updating, 50 samples per example, and 25 training iterations.

- **SampleRank-Multi**: This uses the same configuration as **SampleRank** above, but the Gibbs sampler is replaced by the Multi-Task Metropolis-Hastings sampler described in the previous section. Similarly, a 0-temperature version of the sampler is used for prediction.

For comparison, we also compare POS accuracy with **CRF**, a first-order CRF trained independently on the POS data; and we compare NER accuracy with **upper-bound**, which is a linear-chain CRF using the *true POS labels* for each testing sentence.

9.3.4.3 Results

We evaluate POS accuracy, NER accuracy, as well as the joint accuracy over both tasks. Figures 9.7, 9.8, and 9.9 display results for POS, NER, and their combination as the number of training iterations of SampleRank increases. All differences after 25 iterations are statistically significant according to McNemar's test except for the difference in NER accuracy between **SampleRank** and **BeamSearch** in Figure 9.8.

The first results is negative: According to Figure 9.7, jointly maximizing POS and NER actually *decreases* the accuracy of POS. The reason for this in the **BeamSearch** system is most likely the fact that the probabilities output by each system will differ in register. That is, the probability of the best scoring POS label sequence will most likely be smaller than the probability of the best scoring NER label sequence, simply because the output space of the POS task is larger. Thus, maximizing the product of these two probabilities will likely sacrifice POS accuracy for NER accuracy.

This negative result also holds to a lesser extent for the SampleRank systems. The same argument does not hold here, since SampleRank jointly learns the parameters for each task, and so the scores should be in the same register. Another possible cause is that the larger output space of the POS task requires more attention by the prediction algorithm. For instance, the prediction algorithm may need to draw more samples from the POS task to attain a comparable level of accuracy as for the NER tasks.

The second result is that **SampleRank-Multi** performs significantly better than **SampleRank** and **BeamSearch** when measured for POS accuracy, NER accuracy, or joint POS and NER. This results suggests that more sophisticated jump functions are useful for more efficient exploration of the output space in multi-task inference.

9.4 Related Work

Developing efficient algorithms for boolean satisfiability is an extremely active area of research [55]. Existing algorithms for SAT can broadly be categorized as *complete* or *incomplete*. Complete solvers are guaranteed to find a satisfying assignment if one exists. Almost all complete algorithms are variants of the original Davis-Putnam-Logemann-Loveland (DPLL) algorithm, a recursive backtracking search method enhanced with unit propagation and literal subsumption [28]. Improvements to this algorithm include more efficient branching heuristics [12], data structures [118], constraint propagation [75], and backtracking [97].

Incomplete SAT algorithms are generally variants of local search methods. WalkSat is a typical example [98], which performs a random walk, alternating between greedy and random variable assignments. Complete and incomplete algorithms for SAT have been extended to solve MAX-SAT and WMAX-SAT [49, 119, 45, 8].

Related work on decomposing maximization problems include Dechter and Pearl [30], who propose tree-based decompositions for constraint networks; Bjesse et al. [7], who propose tree-based decompositions for (unweighted) SAT problems; and Amir and McIlraith [1], who propose message passing algorithms for first-order logic. These approaches consider neither weighted maximization problems nor sparse message computation. Additionally, Braunstein et al. [10] recently proposed an algorithm for unweighted SAT called *survey propagation*. Survey propagation can be understood as a variant of traditional (non-generalized) belief propagation, and therefore may not be well-suited for clustered SAT instances.

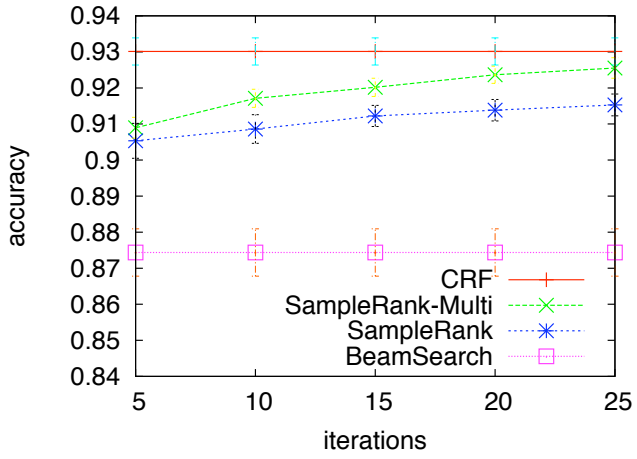

Figure 9.7. POS accuracy on the seminars data.

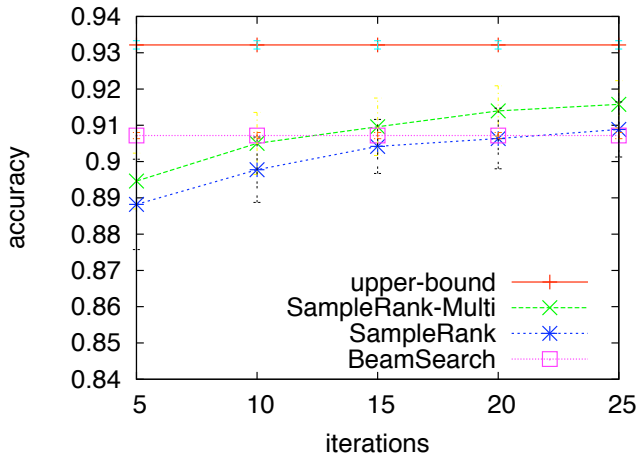

Figure 9.8. NER accuracy on the seminars data.

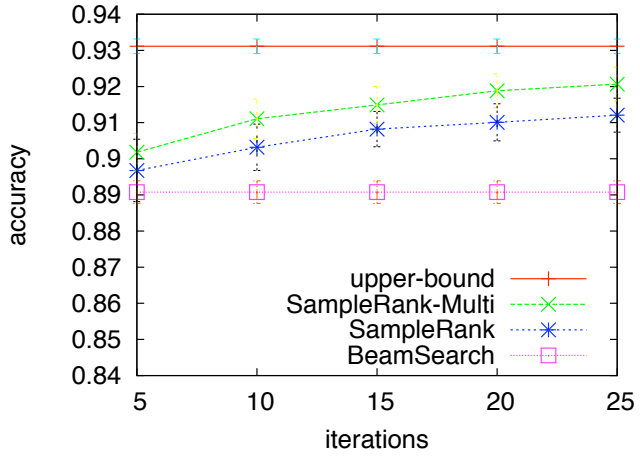

Figure 9.9. Joint POS and NER accuracy on the seminars data.

Sparse message passing has been studied previously, though mainly for real-valued domains [59, 103]. For example, Sudderth et al. [103] define a message as a kernelized mixture of Gaussians, which are sampled using Gibbs sampling. Related work in discrete domains include Pal et al. [81], who introduce a learning algorithm for linear-chain models with large state spaces based on a variant of beam-search; and Duchi et al. [33], who propose combining dynamic programming with message passing using univariate messages.

Sutton et al. [108] present a *factorial CRF* for pipelines of language tasks. While this approach is shown to perform well on several language problems, its principal limitation is that the dependencies across problems must be factored in order to make (even approximate) inference scalable. In contrast, our approach is able to model arbitrary, long-range dependencies between problems because the prediction for one problem is converted to observations for another problem.

Finkel et al. [36] present a Bayesian network for pipelines of language tasks where each variable in the network corresponds to a subtask. Inference proceeds by sampling from each variable in topological order. This approach is attractive for its simplicity (little modification of existing inference algorithms is required); however, because the graphical model is directed, it is difficult in this approach to enable predictions for later problems to influence the predictions for earlier problems.

Poon and Domingos [89] present a Markov logic network for joint segmentation and deduplication of citation records. Their model can be understood as a single global model that uses approximate learning and inference methods. In this paper, we have shown that leveraging exact inference methods for subtasks can enhance performance of such global models.

Wellner et al. [115] present a conditional random field for joint segmentation and deduplications. Communication between tasks is accomplished by passing k-best lists of solution, and no non-local features are used across tasks.

Felzenszwalb and McAllester [34] propose a compositional model of multitask structured prediction, where the prediction is equated to a lightest derivation problem. Their approach, however, does not allow global features over the output space.

One way of understanding Multi-Task Metropolis-Hastings is as in instance of Block Gibbs sampling [42] with the addition of the proposal distribution constructed from the local models.

CHAPTER 10

FUTURE WORK

In this section, we outline a few proposals for future research directions.

10.1 Inference

An important direction of future work is to evaluate probabilistic inference (in addition to the evaluation of prediction performed here). Because we resorted to greedy versions of inference algorithms to perform prediction, our approach is vulnerable to local minima. As we consider more complex tasks, this will likely become increasingly important.

While the inference methods proposed here aimed to be as memory-efficient as possible, in the end they only reason over propositional representations. An interesting avenue of future work is to combine ideas from lifted inference [87, 29] to create a hybrid algorithm that performs some reasoning at the first-order level and only propositionalizes when necessary.

Another avenue of research concerns constructing a formal language to specify proposal distributions for Metropolis-Hastings. While the types of domain knowledge needed for constructing a proposal distribution seems no more demanding than that used for feature engineering, actually defining valid transitions may require more engineering than is desirable. Instead, it would be interesting to investigate ways to express valid solutions and transitions between solutions in weighted logic. This may lead to interesting theoretical issues to distinguish classes of transitions that can be computed and verified in polynomial time versus those that require exponential time.

Regarding multi-task inference, we feel we have only begun to address the problem of sharing partial solutions across tasks. In terms of the message passing algorithms we proposed, we plan to explore more sophisticated sparse message representations (e.g., Drechsler and Sieling [32]). Better message compression has the potential to improve both efficiency and accuracy of prediction since each message will represent a greater number of solutions. We also plan to investigate *coarse-to-fine* approaches to message passing, in which messages are compressed by reducing the solution space that they represent. To this end, we are currently considering dimensionality reduction techniques for messages.

Additionally, we would like to consider more theoretical analysis of the message-passing algorithm. Ihler et al. [53] present theoretical work to understand how particular types of message errors affect accuracy. We plan to extend this analysis to bound accuracy based on different types of message compression.

In terms of applications, we plan and to apply our algorithms to longer pipelines of real-world NLP problems, such as speech recognition, parsing, and extraction. Moving toward such holistic systems will both test the scalability of our approach and give us a better sense of the impact of combined top-down and bottom-up processing.

10.2 Learning

We would like to develop a more formal framework to understand SampleRank and related online algorithms. In particular, it seems that the choice of which samples to use as training examples has a significant impact on performance. We have tried to answer this question empirically in the experiments in this thesis, but there may be theoretical approaches to determining appropriate schedules and update types for online learning.

Additionally, related to the idea in the previous section, it is interesting to consider a learning algorithm that can operate with a prediction algorithm that interleaves lifted and propositional inference.

It is also important to consider modifications to make SampleRank less computationally intensive. Generating samples over training examples many times can be time consuming. In the experiments presented here, we have used a fixed number of SampleRank iterations. Instead, we could consider estimating the convergence of the algorithm to reduce the total number of iterations. Alternately, it is possible that a more intelligent ordering of training examples could improve convergence rates – e.g., performing more iterations on harder examples.

CHAPTER 11

CONCLUSION

In this thesis, we have presented and explored key computational issues in parameter learning and inference for weighted logic representations. We have presented MCMC-based inference algorithms that result in improved memory efficiency for many problems, and have proposed two extensions to enable inference over multiple tasks simultaneously.

We have also presented SampleRank, an online learning algorithm designed for the sampling algorithms used for inference. The main computational advantage of SampleRank is that updates are made after each sample, rather than waiting for an estimate of the best prediction. Somewhat surprisingly, we have shown that making parameter updates using only pairs of samples can result in quite accurate models, as demonstrated on synthetic data, as well as important information extraction tasks such as named entity recognition and coreference resolution.

From a NLP perspective, this thesis has demonstrated that often it is worth sacrificing exact dynamic programming solutions in order to employ more sophisticated representations. From a relational learning perspective, we have proposed learning and inference methods that avoid explicit instantiations of entire models, thereby making weighted logic more practical for real-world problems.

As machine learning algorithms are applied to larger, real-world problems, online algorithms like SampleRank grow more appealing in terms of scalability and adaptability. As problem complexity grows, it will become increasingly important to accurately model the high order dependencies among system predictions. Weighted logic provides an intuitive and compact representation to model these interactions, and with the learning and

inference algorithms described in this thesis, we have begun making use of this powerful representation to solve real-world problems.

BIBLIOGRAPHY

[1] Eyal Amir and Sheila McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artif. Intell.*, 162(1-2):49–88, 2005.

[2] B. Amit and B. Baldwin. Algorithms for scoring coreference chains. In *Proceedings of MUC7*, 1998.

[3] E. Bensana, M. Lemaitre, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3):293–299, 1999.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[5] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society, series B*, B36:192–236, 1974.

[6] Indrajit Bhattacharya and Lise Getoor. A latent dirichlet model for unsupervised entity resolution. In *SDM*, 2006.

[7] P. Bjesse, J. Kukula, R. Damiano, T. Stanion, and Y. Zhu. Guiding SAT diagnosis with tree decompositions. In *SAT 2003*, 2003.

[8] B. Borchers and J. Furman. A two-phase exact algorithm for max-sat and weighted max-sat problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 1999.

[9] J. Boyan and A. Moore. Using prediction to improve combinatorial optimization search. In *Sixth International Workshop on Artificial Intelligence and Statistics*, 1997.

[10] A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: an algorithm for satisfiability. *Random Structures and Algorithms*, 27:201–226, 2005.

[11] Eric Brill. Some advances in rule-based part of speech tagging. In *AAAI*, 1994.

[12] Renato Bruni and Andrea Santori. Adding a new conflict based branching heuristic in two evolved DPLL SAT solvers. In *The Seventh International Conference on Theory and Applications of Satisfiability Testing*, Vancouver, Canada, May 2004.

[13] Razvan Bunescu and Raymond J. Mooney. Collective information extraction with Relational Markov Networks. In *ACL*, 2004.

[14] Y. Censor and S.A. Zenios. *Parallel optimization: theory, algorithms, and applications*. Oxford University Press, 1997.

[15] M. Chang, Q. Do, and D. Roth. Multilingual dependency parsing: A pipeline approach. In Nicolas Nicolov, editor, *Recent Advances in Natural Language Processing*, pages 195–204. Springer-Verlag, July 2006.

[16] Michael Collins. Discriminative reranking for natural language parsing. In *Proc. 17th International Conf. on Machine Learning*, pages 175–182. Morgan Kaufmann, San Francisco, CA, 2000.

[17] Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

[18] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *ACL*, 2004.

[19] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, March 2006.

[20] Aron Culotta and Andrew McCallum. Tractable learning and inference with high-order representations. In *International Conference on Machine Learning Workshop on Open Problems in Statistical Relational Learning*, Pittsburgh, PA, 2006.

[21] Aron Culotta, Pallika Kanani, Robert Hall, Michael Wick, and Andrew McCallum. Author disambiguation using error-driven machine learning with a ranking loss function. In *Sixth International Workshop on Information Integration on the Web (IIWeb-07)*, Vancouver, Canada, 2007.

[22] Aron Culotta, Andrew McCallum, Bart Selman, and Ashish Sabharwal. Sparse message passing algorithms for weighted maximum satisfiability. In *New England Student Colloquium on Artificial Intelligence (NESCAI)*, Ithaca, NY, 2007.

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

[24] James Cussens. Loglinear models for first-order probabilistic reasoning. In Kathryn Blackmond Laskey and Henri Prade, editors, *Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI'99)*, pages 126–133. Morgan Kaufmann, 1999.

[25] Hal Daumé III and Daniel Marcu. A large-scale exploration of effective global features for a joint entity detection and tracking model. In *HLT/EMNLP*, Vancouver, Canada, 2005.

[26] Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*, Bonn, Germany, 2005.

[27] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Submitted to the Machine Learning Journal*, 2006.

[28] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[29] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *IJCAI*, pages 1319–1325, 2005.

[30] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–266, 1989.

[31] Pascal Denis and Jason Baldridge. A ranking approach to pronoun resolution. In *IJCAI*, 2007.

[32] Rolf Drechsler and Detlef Sieling. Binary decision diagrams in theory and practice. *Int J STTT*, 3:112–136, 2001.

[33] J. Duchi, D. Tarlow, G. Elidan, and D. Koller. Using combinatorial optimization within max-product belief propagation. In *Advances in Neural Information Processing Systems (NIPS 2006)*, 2007.

[34] Pedro F. Felzenszwalb and David A. McAllester. The generalized A* architecture. *J. Artif. Intell. Res. (JAIR)*, 29:153–190, 2007.

[35] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *ACL*, pages 363–370, 2005.

[36] Jenny Rose Finkel, Christopher D. Manning, and Andrew Y. Ng. Solving the problem of cascading errors: Approximate Bayesian inference for linguistic annotation pipelines. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2006)*, pages 618–626, 2006.

[37] Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[38] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *Computational Learing Theory*, pages 209–217, 1998.

[39] F. R. Kschischang B. J. Frey and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[40] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[41] H. Gaifman. Concerning measures in first order calculi. *Israel J. Math*, 2:1–18, 1964.

[42] Zoubin Ghahramani, Tommi Jaakkola, Michael Jordan, and Lawrence Saul. An introduction to variational methods in graphical models. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, 1997.

[43] W. Gilks, S. Richardson, and D. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

[44] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, London, 1997.

[45] Michel Goemans and David P. Williamson. Approximation algorithms for max-sat. *SIAM Journal on Discrete Mathematics*, pages 656–666, 1994.

[46] P. Haddawy. Generating bayesian networks from probability logic knowledge bases. In R. Lopez de Mantaras and D. Poole, editors, *Uncertainty In Artificial Intelligence 10 (UAI94)*, pages 262–269. Morgan Kaufmann, 1994.

[47] Aria Haghighi and Dan Klein. Unsupervised coreference resolution in a nonparametric bayesian model. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 848–855, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[48] J. Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.

[49] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[50] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.

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

[52] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

[53] A. T. Ihler, J. W. Fisher III, and A. S. Willsky. Message errors in belief propagation. In *Neural Information Processing Systems*, 2004.

[54] Y. Jiang, H. Kautz, and B. Selman. Solving problems with hard and soft constraints using a stochastic algorithm for max-sat. In *1st Workshop on Artificial Intellignce and Operations Research*, 1995.

[55] Henry Kautz and Bart Selman. The state of sat. *Discrete and Applied Math*, 2006.

[56] Kristian Kersting and Luc De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.

[57] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[58] Daphne Koller and Avi Pfeffer. Probabilistic frame-based systems. In *In Proc. 15th AAAI National Conference on Articial Intelligence*, pages 580–587, 1998.

[59] Daphne Koller, Uri Lerner, and Dragomir Angelov. A general algorithm for approximate inference and its application to hybrid bayes nets. In *UAI*, 1999.

[60] Peter Koomen, Vasin Punyakanok, Dan Roth, and Wen T. Yih. Generalized inference with multiple semantic role labeling systems. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 181–184, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W05/W05-0625`.

[61] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[62] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Roy. Stat. Soc.*, B50, 1988.

[63] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45(3, (Ser. B)):503–528, 1989.

[64] Daniel Lowd and Pedro Domingos. Efficient weight learning for Markov Logic Networks. In *Proceedings of the Eleventh European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 200–211, Warsaw, Poland, 2007. Springer.

[65] Chris Manning and Hinrich Schtze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, May 1999.

[66] Bhaskara Marthi, Brian Milch, and Stuart Russell. First-order probabilistic models for information extraction. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 71–78, Acapulco, Mexico, August 2003.

[67] A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *IJCAI Workshop on Information Integration on the Web*, 2003.

[68] Andrew McCallum and Ben Wellner. Conditional models of identity uncertainty with application to noun coreference. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *NIPS17*. MIT Press, Cambridge, MA, 2005.

[69] R. McDonald and F. Pereira. Online learning of approximate dependency parsing algorithms. In *European Association for Computational Linguistics (EACL)*, 2006.

[70] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[71] Brian Milch. *Probabilistic Models with Unknown Objects*. PhD thesis, Computer Science Division, University of California, Berkeley, December 2006.

[72] Brian Milch, Bhaskara Marthi, and Stuart Russell. BLOG: Relational modeling with unknown objects. In *ICML 2004 Workshop on Statistical Relational Learning and Its Connections to Other Fields*, 2004.

[73] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic models with unknown objects. In *IJCAI*, 2005.

[74] R. Moll, A. Barto, T. Perkins, and R. Sutton. Learning instance-independent value functions to enhance local search. In *NIPS*, 1998.

[75] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, , and Sharad Malik. CHAFF: Engineering an Efficient SAT Solver. In *Design Automation Conference*, 2001.

[76] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, 1995.

[77] R. M. Neal. Probabilistic inference using Markov Chain Monte Carlo methods. Technical report, Dept. of Computer Science, University of Toronto, 1993.

[78] Jennifer Neville and David Jensen. Dependency networks for relational data. In *ICDM*, pages 170–177, 2004.

[79] Vincent Ng. Machine learning for coreference resolution: From local classification to global ranking. In *ACL*, 2005.

[80] Vincent Ng and Claire Cardie. Improving machine learning approaches to coreference resolution. In *ACL*, 2002.

[81] Chris Pal, Charles Sutton, and Andrew McCallum. Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2006.

[82] James D. Park. Using weighted max-sat engines to solve mpe. In *AAAI/IAAI*, pages 682–687, 2002.

[83] Mark A. Paskin. Maximum entropy probabilistic logic. Technical Report UCB/CSD-01-1161, University of California, Berkeley, 2002.

[84] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.

[85] Fuchun Peng and Andrew McCallum. Accurate information extraction from research papers using conditional random fields. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 329–336, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

[86] A. Pfeffer. IBAL: An integrated bayesian agent language. In *IJCAI*, 2001.

[87] D. Poole. First-order probabilistic inference. In *IJCAI*, pages 985–991, Acapulco, Mexico, 2003. Morgan Kaufman.

[88] Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 458–463, Boston, MA, 2006. AAAI Press.

[89] Hoifung Poon and Pedro Domingos. Joint inference in information extraction. In *AAAI*, pages 913–918, 2007.

[90] Hoifung Poon, Pedro Domingos, and Marc Sumner. A general method for reducing the complexity of relational inference and its application to mcmc. In *AAAI*, 2008.

[91] L.R. Rabiner. A tutorial on hidden Markov models. In *IEEE*, volume 77, pages 257–286, 1989.

[92] Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175, 1999.

[93] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[94] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer, July 2005.

[95] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[96] D. Roth and W. Yih. A linear programming formulation for global inference in natural language tasks. In *The 8th Conference on Compuational Natural Language Learning*, May 2004.

[97] Tian Sang, Fahiem Bacchus, Paul Beame, Henry Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *The Seventh International Conference on Theory and Applications of Satisfiability Testing*, Vancouver, Canada, May 2004.

[98] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *Proceedings 1993 DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[99] Parag Singla and Pedro Domingos. Memory-efficient inference in relational domains. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston, MA, 2006.

[100] Wee Meng Soon, Hwee Tou Ng, and Daniel Chung Yong Lim. A machine learning approach to coreference resolution of noun phrases. *Comput. Linguist.*, 27(4):521–544, 2001. ISSN 0891-2017.

[101] Stephanie Strassel, Alexis Mitchell, and Shudong Huang. Multilingual resources for entity extraction. In *Proceedings of the ACL 2003 workshop on Multilingual and mixed-language named entity recognition*, pages 49–56, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[102] T. Stützle, H. Hoos, and A. Roli. A review of the literature on local search algorithms for max-sat. Technical Report AIDA-01-02, Technische Universität Darmstadt, 2002.

[103] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. In *CVPR*, 2003.

[104] Charles Sutton and Andrew McCallum. Collective segmentation and labeling of distant entities in information extraction. In *In ICML Workshop on Statistical Relational Learning and Its Connections to Other Fields*, 2004.

[105] Charles Sutton and Andrew McCallum. Joint parsing and semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 225–228, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[106] Charles Sutton and Andrew McCallum. Piecewise training of undirected models. In *21st Conference on Uncertainty in Artificial Intelligence*, 2005.

[107] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2006. To appear.

[108] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *J. Mach. Learn. Res.*, 8:693–723, 2007.

[109] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[110] Ben Taskar, Abbeel Pieter, and Daphne Koller. Discriminative probabilistic models for relational data. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighteenth Conference (UAI-2002)*, pages 485–492, San Francisco, CA, 2002. Morgan Kaufmann Publishers.

[111] Francois Trouilleux, Gabriel G. Bes, and Eric Gaussier. An evaluation of inter-annotator agreement in the observation of anaphoric and referential relations. In *Discourse, Anaphora and Reference Resolution Conference*, 2000.

[112] Joseph Turian, Benjamin Wellington, and I. Dan Melamed. Scalable discriminative learning for natural language parsing and translation. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1409–1416. MIT Press, Cambridge, MA, 2007.

[113] Joshua R. Tyler, Dennis M. Wilkinson, and Bernardo A. Huberman. Email as spectroscopy: Automated discovery of community structure within organizations. Technical report, Hewlett-Packard Labs, 2003.

[114] Y. Weiss and W. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47, 2001.

[115] Ben Wellner, Andrew McCallum, Fuchun Peng, and Michael Hay. An integrated, conditional model of information extraction and coreference with application to citation matching. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[116] H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proc. of the 8th Intern. Workshop on Parsing Technologies*, pages 195–206, 2003.

[117] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *NIPS*, pages 689–695, 2000.

[118] Hantao Zhang and Mark E. Stickel. Implementing the Davis-Putnam Method. *Journal of Automated Reasoning*, 2000.

[119] Hantao Zhang, Haiou Shen, and Felip Many. Exact algorithms for MAX-SAT. *Electr. Notes Theor. Comput. Sci*, 86, 2003.

[120] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intellience*, 1995.