Inference and Learning in Large Factor Graphs with Adaptive Proposal Distributions and a Rank-based Objective

Khashayar Rohanimanesh, Michael Wick, and Andrew McCallum

CMPSCI Technical Report
UM-CS-2009-008

May 26th, 2009 Department of Computer Science

University of Massachusetts
140 Governors Drive
Amherst, Massachusetts 01003

Abstract

Large templated factor graphs with complex structure that changes during inference have been shown to provide state-of-the-art experimental results in tasks such as identity uncertainty and information integration. However, inference and learning in these models is notoriously difficult. This paper formalizes, analyzes and proves convergence for the SampleRank algorithm, which learns extremely efficiently by calculating approximate parameter estimation gradients from each proposed MCMC jump. Next we present a parameterized, adaptive proposal distribution, which greatly increases the number of accepted jumps. We combine these methods in experiments on a real-world information extraction problem and demonstrate that the adaptive proposal distribution requires 27% fewer jumps than a more traditional proposer.

1 Introduction

Relational factor graphs are a popular framework for modeling structured prediction problems where there are dependencies amongst the hidden variables. Some of the first applications, such as sequence labeling, exploited the linear-chain factorization of the graph to efficiently perform exact structured prediction. More recently however, the expressive power of probabilistic programming languages [16, 14, 7, 13] have given rise to graphs with notoriously difficult learning and inference problems. The purpose of this work is to propose and analyze an approach to learning and inference that empirically has produced state-of-the-art results.

First we consider the problem of learning. Traditionally, gradient based methods (e.g., maximum-likelihood method) have been used for learning parameters in simple factor graphs. However, computing gradients typically involves expensive inference routines; for example, maximum-likelihood requires computing marginals. The problem with these methods is that the inference step is performed inside the inner loop of parameter updates of learning (between each gradient update). A number of approaches address this issue to various degrees with different approximations to the gradient [1, 2, 10, 9]. For example, Collin's perceptron avoids marginal computations, but requires decoding; contrastive divergence approximates marginals with sampling; and LASO scores partial configurations based on whether or not they could lead to the ground truth. On the same thread, the SampleRank algorithm computes gradients between complete (but incorrect) neighboring configurations from a chain induced by Markov-Chain Monte Carlo (MCMC) [3]. There are three advantages to this approach (1) parameter updates are in the inner-most loop, avoiding inference at each step; (2) the resulting objective function has more constraints than maximum likelihood which may prevent over-fitting; (3) any configuration pair is a potential training example taking advantage of more data.

Previous work has demonstrated the empirical success of SampleRank; for example, [4] acheives state of the art results on an information extraction task; however, there has been no theoretical analysis of SampleRank's properties. One of the contributions of our paper is that we establish the convergence of SampleRank for the first time, and derive the necessary conditions for the algorithm to realize a rank-based objective. A corollary of the proof is that SampleRank does not require strict MCMC to converge, allowing the use of alternative inference methods (such as local search).

At the heart of the SampleRank algorithm is a jump function that induces a random walk through the configuration space; in practice, however, the performance of the algorithm depends significantly on the quality of these jumps. A second contribution of this work is the development of a novel approach for learning a parameterized jump function that uses the cross entropy method (CEM) [5, 6] to shift the sampling distribution toward the true model.

We also present some initial results showing that the learned jump function can guide the algorithm to the goal configuration using 27% fewer inference steps when compared to its non-adaptive counterpart.

The rest of this paper is organized as follows: in Section 2 we present an overview of relational CRFs and the cross entropy method. In Section 3 we outline the proof of convergence for the SampleRank algorithm. In Section 4 we introduce a new class of adaptive proposers based on the cross entropy method. In Section 5 we present some experimental results. Finally in Section 6 we conclude and outline a few directions for future work.

2 Preliminaries

This section serves to provide a brief overview of relational CRFs and the cross entropy method (CEM). For clarity, we first introduce the notation used throughout the paper. We use upper case bold-faced notation to represent collections of variables, and lower case bold-faced notation to represent individual variables (for example $\mathbf{X} = \{\mathbf{x}_i, \mathbf{x}_{ij}\}$, $\mathbf{Y} = \{\mathbf{y}_i, \mathbf{y}_{ij}\}$, etc). We use non-bold-faced notation for a particular assignment to the variables (for example $X = \{x_i, x_{ij}\}$, $Y = \{y_i, y_{ij}\}$, etc). Depending on the context, when we refer to a particular instantiation of variables, we use subscript indexing (e.g., Y_k, Y_t), however for clarity the index will appear as a superscript in the individual variables (for example $Y_t = \{y_i^t, y_{ij}^t\}$). We also use $\mathcal{X}$ to denote the set of individual objects (or observations) in the world, whereas an X_i denotes a particular partition of $\mathcal{X}$. For simplicity we only use $\mathcal{X}$ whenever a particular partitioning of the inputs is implied.

2.1 Relational CRFs

We briefly overview some of the basic definitions and concepts in relational CRFs (for more details see [3]). A conditional random field (CRF) [12] is a discriminative factor graph that gives a conditional probability distribution over an assignment to a set of hidden variables $\mathbf{Y} = Y$ given a set of observed variables $\mathbf{X} = X$. However, in relational CRFs, an assignment to the hidden variables denotes a particular hypothesis (e.g., a relation, a property, etc) over the observables. This is further illustrated in Figure 1 which shows a partially instantiated factor graph for a generic clustering problem. Observations are indicated by the geometric shapes, and the dotted circles show the current clustering (hypothesis). In general, the hidden variables encode various relationships among the observations: for example variables $\mathbf{y}_i$s indicate whether or not a relationship within a cluster of observations (i.e., x_is) holds. Other types of hidden variables can also be defined to

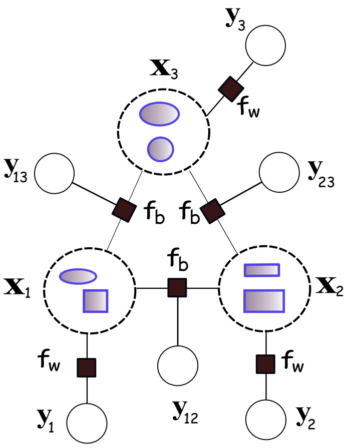

Figure 1: Factor graph representation of a relational CRF (partially instantiated).

capture additional relationships; for example, variables $\mathbf{y}_{ij}s$ indicate some compatibility between a pair of clusters of observations (i.e., x_i and x_j) [1].

Next, the conditional probability distribution $P(Y|X)$ is computed as:

$$\mathcal{P}(\mathbf{Y} = Y|\mathbf{X} = X, \Theta) = \\ \frac{1}{\mathcal{Z}_X} \prod_{y_i \in Y} f_w(y_i, x_i) \prod_{y_i, y_j \in Y} f_b(y_{ij}, x_{ij}) \quad (1)$$

where $\mathcal{Z}_{\mathbf{X}}$ is the input-dependent normalizer, factor f_w parameterizes the *within-cluster* compatibility, and factor f_b parameterizes the *between-cluster* compatibility[2]. Traditionally, a log-linear model of potential functions (i.e., f_w and f_b) is employed given the model parameter Θ and a set of features $\phi = \{\phi_k\}$ [3]:

$$f_w(y_i, x_i) = \exp\left(\sum_k \Theta_k \, \phi_k(x_i, y_i)\right)$$

A desirable facet of our model is that it factorizes into clusters of data rather than pairs (Equation 1). This enables us to define features of entire clusters using *first-order logic features*: features that can universally and existentially aggregate properties of a set of objects. This model can be intuitively described as follows: every possible clustering of the data induces a different set of instantiations of $\mathbf{Y}$ variables and gives different assignments to them. The conditional distribution $\mathcal{P}(\mathbf{Y} = Y|\mathbf{X} = X, \Theta)$ gives the probability of a configuration $\mathbf{Y} = Y$ measured in terms of

[1] In general the inference and learning methods could be applied to a variety of model structures.

[2] In the above equation we use the notation x_{ij} to denote a pair of clusters x_i and x_j.

[3] Here we just show the within cluster factor representations (e.g., f_ws), the between cluster factors (e.g., f_bs) can be similarly represented.

Algorithm 1 SR-Learn

```
Inputs:
        $\mathcal{M} \equiv \langle \mathcal{X}, \phi, \Theta \rangle$: model
        $Y_{t+1} \leftarrow \mathcal{J}(Y_t, \mathcal{X}; \lambda)$: (parametrized) proposer
        $\mathcal{F}(Y)$: Performance metric (e.g., F1 metric)
        $Y_0$: Initial state
Initialization:
Set $\bar{\Theta} = \mathbf{0}$
Output:
Parameter vector $\bar{\Theta}$
for $t = 1, \ldots, T$ do
    Generate a training instance:
            $Y_{t+1} \leftarrow \mathcal{J}(Y_t, \mathcal{X}; \lambda)$
    Compute:
            $Y^+ = \mathrm{argmax}_{Y \in \{Y_i, Y_{t+1}\}} \mathcal{F}(Y)$   and
            $Y^- = \arg min_{Y \in \{Y_t, Y_{t+1}\}} \mathcal{F}(Y)$
    Let: Score$^{\Theta}_{\mathcal{X}}(Y) = \Theta \,.\, \phi(\mathcal{X}, Y)$
    if Score$^{\bar{\Theta}}_{\mathcal{X}}(Y^+) -$ Score$^{\bar{\Theta}}_{\mathcal{X}}(Y^-) < 0$ then
        $\bar{\Theta} = \bar{\Theta} + \phi(\mathcal{X}, Y^+) - \phi(\mathcal{X}, Y^-)$
    end if
end for
```

a normalized score of how likely that configuration is. We parameterize this score with a set of potential functions that evaluate the compatibility of both within-cluster attributes and between-cluster attributes.

Maximum-likelihood estimate of the parameter gives:

$$\frac{\partial}{\partial \Theta_k} L(Y; X, \Theta) = E_{\mathcal{D}}[\phi_k] - E_{\mathcal{M}}[\phi_k]$$

where $E_{\mathcal{D}}[g_k]$ denotes the expected value of some feature g_k with respect to the empirical distribution in training data $\mathcal{D}$, and $E_{\mathcal{M}}[g_k]$ denotes the expected value of the feature g_k with respect to the distribution given by the model (parametrized by Θ). The second term in the above equation involves computing model expectations over features, requiring a summation over all hypothesis Y which is intractable in relational CRFs.

[3] introduced the *SampleRank* algorithm that incorporates *perceptron*-style updates while performing a random walk in the hypothesis space (i.e., $\mathbf{Y}$) in order to learn a model that agrees with the correct ranking of the hypotheses (according to the ground truth). The proposed algorithm assumes MAP inference is performed with Markov Chain Monte-Carlo and then updates parameters with approximate gradients at each MCMC step. More specifically, each step of MCMC induces

a neighbor configuration pair by modifying a hypothesis Y to produce a new hypothesis Y'. If the number of $\mathbf{y} \in \mathbf{Y}$ variables modified is a constant, then the difference between Y' and Y is small and computing their gradient is efficient (in practice this often involves computing just a constant number of factors, for more details refer to [3]).

Algorithm 1 shows the detail of the SampleRank algorithm. The system is initialized at a random hypothesis Y_0 (for example by placing every observation in single cluster). At every iteration of the algorithm, a proposal distribution produces samples $Y_{t+1} \leftarrow \mathcal{J}(Y_t, \mathcal{X}; \lambda)$ by stochastically proposing (small) modifications to the current hypothesis (for example it could propose to randomly split a cluster, or randomly merge to clusters). Next, a perceptron style update on model parameters is performed if and only if the model does not correctly rank the current and new hypotheses according to the ground truth (e.g., the performance metric $\mathcal{F}$). Once the model is learned, the most plausible hypothesis can be found by simply hill climbing the learned rank function given by the model. The convergence of the SampleRank algorithm has not been established previously. In Section 3 we derive a set of conditions and establish the convergence of the SampleRank algorithm under these conditions.

2.2 The Cross-Entropy Method

In this section we present an overview of the cross entropy method (CEM) which is a powerful method for solving rare event simulation (RES), and combinatorial optimization problems (COPs). For a complete discussion we refer the reader to [5, 6]. In this approach, the original optimization problem is first transformed into an associated stochastic problem, and then an iterative adaptive algorithm is used to solve it. In brief, the CEM iterative algorithm consists of two steps: (1) generate a set of samples (hypothesis, sample solutions, etc) according to some parameterized random mechanism; (2) update the parameters of the random process that is responsible for generating samples on the basis of the performance of the generated samples in order to produce better samples in the next round. To further illustrate this, consider an optimization problem where we wish to maximize some performance function $\mathcal{F}(\mathbf{Y})$ over samples of $\mathbf{Y}$:

$$\gamma^* = \max_{\mathbf{Y}} \mathcal{F}(\mathbf{Y})$$

First, the above deterministic optimization is transformed into a RES by introducing a family of parametrized pdfs $\{\mathcal{J}(\mathbf{Y}; \lambda) | \lambda \in \Lambda\}$ as follows:

$$l(\gamma) = \mathcal{P}_\lambda(\mathcal{F}(Y) \geq \gamma) = E_\lambda[\mathcal{I}_{\{\mathcal{F}(Y) \geq \gamma\}}]$$

where λ is some parameter governing the distribution induced by $\mathcal{J}(\mathbf{Y}; \lambda)$. We can think of the event of "maximum score" as the rare event of our interest in this RES problem. Next, CEM iteratively generates a sequence of pair of estimates $\{\langle\hat{\gamma}_1, \hat{\lambda}_1\rangle, \langle\hat{\gamma}_2, \hat{\lambda}_2\rangle, \ldots, \langle\hat{\gamma}^*, \hat{\lambda}^*\rangle\}$ which converges to very tight neighborhood of the optimal estimates $\langle\hat{\gamma}^*, \hat{\lambda}^*\rangle$.

Algorithm 2 outlines the detail of this algorithm (borrowed from [6]). The parametrized proposal distribution $\mathcal{J}(\mathbf{Y}; \lambda)$ is initialized by a random parameter λ_0. at every iteration t of the

Algorithm 2 CEM-Optimize (Cross Entropy Method)

```
1:  Input: $\mathcal{J}(\mathbf{Y};\lambda)$: Parametrized proposal distribution
           $\mathcal{F}(Y)$: Performance metric (e.g., F1 metric)
           $\rho$: Percentage of top samples to keep
2:  Initialization set $\lambda_0 = \bar{0}$
3:  Outputs: Parameter vector $\lambda^*$, $Y^*$
4:  while not converged do
5:    Generate a set of $N$ samples:
           $\{Y_1, Y_2, \ldots, Y_N\} \sim \mathcal{J}(\mathbf{Y};\lambda_t)$
6:    Sort samples according to the performance metric $\mathcal{F}(Y_i)$ in ascending order:
           $\{\mathcal{F}(1), \mathcal{F}(2), \ldots, \mathcal{F}(N)\}$
7:    let $\hat{\gamma_t} = \mathcal{F}(\lceil \rho N \rceil)$
8:    Solve:
      $\bar{\lambda} = \arg max_\lambda \, E_{\lambda_{t-1}}[\mathcal{I}_{\{\mathcal{F}(Y)\geq\hat{\gamma_t}\}} \log \mathcal{J}(Y;\lambda)]$
9:    $\lambda_{t+1} = \bar{\lambda}$
10: end while
```

algorithm, a set of N samples (where the number N is user defined) are generated from $\mathcal{J}(\mathbf{Y};\lambda_t)$. Next these samples are sorted according to the performance metric $\mathcal{F}(\mathbf{Y})$, and a top-portion of these samples (specified by the user defined percentage ρ) are selected for updating the parameter of the proposal distribution for the next round. In order to to update the parameters CEM requires solving an internal optimization (the equation in line 8 of the Algorithm 2). The stochastic counterpart of this optimization can be casted as:

$$\bar{\lambda} = \arg max_\lambda \, \frac{1}{N} \sum_{k=1}^{N} [\mathcal{I}_{\{\mathcal{F}(Y_k)\geq\hat{\gamma_t}\}} \log \mathcal{J}(Y_k;\lambda)] \tag{2}$$

[6] demonstrated that when $\mathcal{J}(\mathbf{Y};\lambda)$ belongs to Natural Exponential Family (e.g., Gaussian, discrete Bernoulli), then closed form solution can be obtained for Equation 2. In general, however it may not be trivial to derive a closed for solution and thus we may rely on numerical methods in order to find an approximate solution.

In practice, whenever a new parameter $\bar{\lambda}$ is estimated by solving Equation 2 we may use an *smoothed* version, where we replace the line 9 of the Algorithm 2 by:

$$\lambda_{t+1} = \alpha \, \bar{\lambda} + (1-\alpha) \, \lambda_t$$

In Section 4 we describe an approach based on CEM for optimizing the performance of family of proposal distributions for generating more plausible jumps in relational CRFs when used in SampleRank algorithm.

3 Convergence Results

In this section we sketch a proof of convergence for the SampleRank algorithm.

Definition 1 For an input $\mathcal{X}$, let $\mathcal{Y}(\mathcal{X})$ be a function that enumerates the set of all possible structured outputs Y_i. Also, let $\mathcal{D}_Y$ denote the set of $Y_i \in \mathcal{Y}(\mathcal{X})$ involved in a training data set $\mathcal{D} = \{Y_i|\mathcal{X}\}_{i=1}^n$ (note that the input observations X_i that are partitions of $\mathcal{X}$ are implied by Y_i).

Definition 2 For any $Y \in \mathcal{D}_Y$, define:

$$\mathcal{L}_{\mathcal{X}}^{<}(Y) = \{Y^- \in \mathcal{D}_Y \,|\, \mathcal{F}(Y^-) < \mathcal{F}(Y)\} \tag{3}$$

Definition 3 For any instance Y and parameter vector Θ, define:

$$\text{Score}_{\mathcal{X}}^{\Theta}(Y) = \Theta\,.\,\phi(\mathcal{X}, Y) \tag{4}$$

Definition 4 A training set $\mathcal{D} = \{Y_i|\mathcal{X}\}_{i=1}^n$ is called *separable with margin* $\delta > 0$ if there exists some vector with $\parallel U \parallel = 1$ such that:

$$\forall Y_i, \forall \bar{Y} \in \mathcal{L}_{\mathcal{X}}^{<}(Y_i),\ \text{Score}_{\mathcal{X}}^{U}(Y_i) - \text{Score}_{\mathcal{X}}^{U}(\bar{Y}) \geq \delta \tag{5}$$

Definition 5 We say that the SampleRank algorithm (Algorithm 1) with parameter vector Θ makes an *error* on a training instance Y_i if $\exists \bar{Y} \in \mathcal{L}_{\mathcal{X}}^{<}(Y_i)$, such that $\text{Score}_{\mathcal{X}}^{\Theta}(Y_i) - \text{Score}_{\mathcal{X}}^{\Theta}(\bar{Y}) < 0$.

Theorem 1 *For any training set $\mathcal{D} = \{Y_i|\mathcal{X}\}_{i=1}^n$ generated by Algorithm 1 that is separable with margin δ, for any value of T, then for the SampleRank algorithm (Algorithm 1):*

$$\mathcal{N}_e \leq \frac{R^2}{\delta^2}$$

where R is a constant such that $\forall i, \forall \bar{Y} \in \mathcal{L}_{\mathcal{X}}^{<}(Y_i)$:

$$\parallel \phi(\mathcal{X}, Y_i) - \phi(\mathcal{X}, \bar{Y}) \parallel \leq R$$

and $\mathcal{N}_e$ is the number of errors made (according to the Definition 5) during training.

Proof: We build on the proof of convergence of perceptron algorithm in [1]. Let $\bar{\Theta}^k$ be the parameter vector before the k^{th} error is made at some i^{th} example, where $\bar{\Theta}^{-1} = 0$. Now, let $Y_{i+1} = \mathcal{J}(Y_i, \mathcal{X}; \lambda)$ and let $Y^+ = \arg max_{Y \in \{Y_i, Y_{i+1}\}} \mathcal{F}(Y)$ and $Y^- = \arg min_{Y \in \{Y_i, Y_{i+1}\}} \mathcal{F}(Y)$. This implies that $Y^- \in \mathcal{L}_{\mathcal{X}}^{<}(Y^+)$. Since this training instance incurs an error, it must have been the case that :

$$\text{Score}_{\mathcal{X}}^{\bar{\Theta}^k}(Y^+) - \text{Score}_{\mathcal{X}}^{\bar{\Theta}^k}(Y^-) < 0 \tag{6}$$

and this causes an update $\bar{\Theta}^{k+1} = \bar{\Theta}^k + \phi(\mathcal{X}, Y^+) - \phi(\mathcal{X}, Y^-)$. Taking the inner product of both sides with $U(\parallel U \parallel = 1)$ yields:

$$\begin{aligned} U.\bar{\Theta}^{k+1} &= U.\bar{\Theta}^k + U.\phi(\mathcal{X}, Y^+) - U.\phi(\mathcal{X}, Y^-) \\ &= U.\bar{\Theta}^k + \text{Score}_{\mathcal{X}}^{U}(Y^+) - \text{Score}_{\mathcal{X}}^{U}(Y^-) \\ &\geq U.\bar{\Theta}^k + \delta \end{aligned}$$

the inequality follows because of the separability assumption in Equation 5. Since $\bar{\Theta}^{-1} = 0$, we have $U.\bar{\Theta}^{-1} = 0$. By induction on k, it follows that for all k, $U.\bar{\Theta}^{k+1} = k\delta$. By *Cauchy–Schwarz* inequality we have $U.\bar{\Theta}^{k+1} \leq \| U \| \| \bar{\Theta}^{k+1} \|$, which gives:

$$\| \bar{\Theta}^{k+1} \| \geq k\delta \tag{7}$$

We also derive an upper bound for $\| \bar{\Theta}^{k+1} \|^2$:

$$\begin{aligned} \| \bar{\Theta}^{k+1} \|^2 = & \ \| \bar{\Theta}^{k} \|^2 + \| \phi(\mathcal{X}, Y^+) - \phi(\mathcal{X}, Y^-) \|^2 + \\ & 2\,\bar{\Theta}^k.(\phi(\mathcal{X}, Y^+) - \phi(\mathcal{X}, Y^-)) \\ = & \ \| \bar{\Theta}^{k} \|^2 + \| \phi(\mathcal{X}, Y^+) - \phi(\mathcal{X}, Y^-) \|^2 + \\ & 2\,(\text{Score}_{\mathcal{X}}^{\bar{\Theta}^k}(Y^+) - \text{Score}_{\mathcal{X}}^{\bar{\Theta}^k}(Y^-)) \\ \leq & \ \| \bar{\Theta}^{k} \|^2 + R^2 \end{aligned}$$

the inequality follows because of the assumption in the Theorem ($\| \phi(\mathcal{X}, Y_i) - \phi(x, \bar{y}) \| \leq R$), and the fact that $\text{Score}_{\mathcal{X}}^{\bar{\Theta}^k}(Y^+) - \text{Score}_{\mathcal{X}}^{\bar{\Theta}^k}(Y^-) < 0$ (Equation 6). By induction on k, it follows that for all k:

$$\| \bar{\Theta}^{k+1} \|^2 \leq kR^2 \tag{8}$$

Combining the lower and upper bounds established in Equations 7 and 8, we obtain:

$$\begin{aligned} k^2\delta^2 \leq \| \bar{\Theta}^{k+1} \| \leq kR^2 \\ \Rightarrow k \leq \frac{R^2}{\delta^2} \end{aligned}$$

Definition 6 Given a training set $\mathcal{D} = \{Y_i | \mathcal{X}\}_{i=1}^n$, for a pair $\langle U, \delta \rangle$, define:

$$\begin{aligned} m_i &= U.\phi(\mathcal{X}, Y_i) - U.\phi(\mathcal{X}, \bar{Y}), \text{ where } \bar{Y} \in \mathcal{L}_{\mathcal{X}}^{<}(Y_i) \\ \epsilon_i &= \max\{0, \delta - m_i\} \\ \mathcal{D}_{\langle U, \delta \rangle} &= \sqrt{\sum_{i=1}^{n} \epsilon_i^2} \end{aligned} \tag{9}$$

Theorem 2 *For any training set $\mathcal{D} = \{Y_i | \mathcal{X}\}_{i=1}^n$, for the first pass over the training set of the SampleRank algorithm (Algorithm 1):*

$$\mathcal{N}_e \leq \min_{\langle U, \delta \rangle} \frac{(R + \mathcal{D}_{\langle U, \delta \rangle})^2}{\delta^2}$$

where R is a constant such that $\forall i, \forall \bar{Y} \in \mathcal{L}_{\mathcal{X}}^{<}(Y_i)$*:*

$$\| \phi(\mathcal{X}, Y_i) - \phi(\mathcal{X}, \bar{Y}) \| \leq R$$

min is taken over $\delta > 0$, $\parallel U \parallel= 1$, and $\mathcal{N}_e$ is the number of errors (according to the Definition 4) made during training.

Proof: The proof is a straight-forward application of the related theorem in [1].

4 Efficient Decoding with the Cross Entropy Method

In a relational CRF, *decoding* refers to the inference problem of finding the most probable hypothesis (setting of the hidden variables) given the observed data. [3] uses Metropolis-Hastings (MH) with annealing in order to identify the hypothesis with the highest score. However, computing the *forward* and *backward* jump-probabilities required for Metropolis-Hastings is not trivial for arbitrary jump functions.

To overcome this problem, we introduce an alternative approach based on CEM that avoids the strict requirement of constructing a valid Markov chain. Our algorithm performs a random walk in the hypothesis space that is guided by a family of parameterized proposal distributions $\mathcal{J}(Y_t, \mathcal{X}; \lambda)$ capable of shifting toward the model's probability distribution. During inference (e.g., over the test domain) the system hill climbs the learned rank objective while adjusting the parameters of the proposal distribution using CEM.

More formally, the decoding problem is defined as:

$$Y^* = \operatorname*{argmax}_{\mathbf{Y}} \mathcal{P}(\mathbf{Y}|\mathcal{X}, \Theta)$$

One immediate idea would be to use CEM to directly solve this problem by introducing some parameterized random mechanism $\mathcal{J}(\mathcal{X}; \lambda)$ to generate sample hypothesis Y, and then use the function **CEM-Optimize**$(\mathcal{J}(\mathcal{X}; \lambda), \text{Score}_{\mathcal{X}}^{\Theta}(), \rho)$ (Algorithm 2) where we use the model score as the performance metric (Equation 4) and ρ is the user defined parameter for the CEM algorithm. However, there are two potential problem with this idea. First, it is not trivial to define a model that generates all possible hypothesis in the feasible region of the hypothesis space. For example, one could suggest using a graph based approach where all vertices are the individual observations and an edge between to vertices v_i and v_j is added with some unknown probability Θ_{ij}. This model does not guarantee generating all feasible hypothesis (for example it does not enforce the *transitivity* property among the objects). Second, the complexity of the set $|\mathbf{Y}| = O(\text{Bell}(|\mathcal{X}|))$ which is still a large space to cope with. Another difficulty with this approach is that scoring samples requires computing all the factors in the instantiated graph pertaining to a particular hypothesis.

Inspired by the fact that computing the difference between neighboring configurations can be done efficiently, we can immediately avoid the aformetioned computational problems. For example, in the clustering view of the model, we can create new clusterings (hypothesis) conditioned on a current clustering by simply *splitting* an existing cluster, or *merging* two existing clusters. The split-merge style proposers have been widely used in MCMC, in particular with the Metropolis-Hastings algorithm [8, 11, 15, 14, 3]. Our approach differs from all of these methods in that our split-merge proposer learns to perform better splits or merges as we take a random walk in the

Algorithm 3 CEM-Decode

```
Inputs:
Model: $\mathcal{M} \equiv \langle \mathcal{X}, \phi, \Theta \rangle$
Proposer: $Y_{t+1} \leftarrow \mathcal{J}(Y_t, \mathcal{X}; \lambda_0)$
Initial State: $Y_0$
Tolerance: $\epsilon > 0$
Output: $Y^* \approx \operatorname{argmax}_{\mathbf{Y}} \Theta \,.\, \phi(\mathcal{X}, \mathbf{Y})$
for $t = 0, \ldots,$ do
  $\lambda_{t+1} \leftarrow$ CEM-Optimize$(\mathcal{J}(Y_t, \mathcal{X}; \lambda_t), \text{Score}^{\Theta}_{\mathcal{X}}(), \rho)$
  {learn the proposer parameter}
  $Y_{t+1} \leftarrow \mathcal{J}(Y_t, \mathcal{X}; \lambda)$ {Generate next state}
  if $\text{Score}^{\Theta}_{\mathcal{X}}(Y_{t+1}) - \text{Score}^{\Theta}_{\mathcal{X}}(Y_t) > \epsilon$ then
    $Y^* \leftarrow Y_{t+1}$
  else
    return $Y^*$
  end if
end for
```

hypothesis space. We can summarize our approach as follows: first we define a parametrized split-merge proposer $Y_{t+1} \leftarrow \mathcal{J}(Y_t, \mathcal{X}; \lambda)$ that generates a new hypothesis Y_{t+1} given some current hypothesis Y_t by splitting a cluster, or merging a pair of clusters in Y_t. Then while taking a walk in the hypothesis space, we optimize proposer's parameters using CEM (using either a learned-model or a ground truth signal as the performance metric). Intuitively, the parameters may be tied allowing splits and merges to generalize across the entire configuration space. Note that the proposer could be parameterized over the same feature space that is used to learn the model.

We now describe one scheme for defining a parametrized split-merge proposer. We use a relational CRF factor graph, where the within cluster factors (i.e., f_ws) and between cluster factors (i.e., f_bs) are parametrized by the parameter vector λ (which serves as the parameter vector for the proposer). It is important to note that this model is totally separate from the model that we learn by the SampleRank algorithm. Then, given some current hypothesis Y_t, we can sample the next hypothesis $Y_{t+1} \sim \mathcal{J}(Y_t, \mathcal{X}; \lambda)$ based on the following mixture model:

$$\begin{aligned} &\mathcal{J}(Y_t, \mathcal{X}; \lambda) = \\ &\beta\, \mathcal{Q}_{\text{split}}(.|Y_t, \mathcal{X}, \lambda) + (1-\beta)\, \mathcal{Q}_{\text{merge}}(.,.|Y_t, \mathcal{X}, \lambda) \end{aligned} \tag{10}$$

where:

$$\begin{aligned} \mathcal{Q}_{\text{split}}(y_i^t|Y_t, \mathcal{X}, \lambda) &= \frac{f_w(y_i^t, x_i^t)}{\sum_{y_j^t \in Y_t} f_w(y_j^t, x_j^t)} \\ \mathcal{Q}_{\text{merge}}(y_i^t, y_j^t|Y_t, \mathcal{X}, \lambda) &= \frac{f_b(y_{ij}^t, x_{ij}^t)}{\sum_{\{y_k^t, y_l^t\} \in Y_t} f_b(y_{kl}^t, x_{kl}^t)} \end{aligned} \tag{11}$$

and $0 \leq \beta \leq 1$ is a fixed mixture parameter that determines how often we prefer to sample the next hypothesis by either splitting a cluster, or merging two clusters. In Equation 11, $\mathcal{Q}_{\text{split}}(y_i^t|Y_t, \mathcal{X}, \lambda)$ gives a normalized measure of within cluster *disparity* of some cluster y_t^i in the current hypothesis Y_t. Similarly $\mathcal{Q}_{\text{merge}}(y_i^t, y_j^t|Y_t, \mathcal{X}, \lambda)$ gives a normalized measure of between cluster *affinity* for a pair of clusters y_i^t and y_j^t in the current hypothesis Y_t. During learning, we would expect that the parameter vector is adjusted so that it produces a higher within cluster disparity score for incorrect clusters, and as a result these clusters get sampled for splitting with a higher frequency. Similarly, we would expect that the learned parameters increase the between cluster affinity score for a pair of clusters (that should be merged) so that get sampled for merges with a higher frequency.

The details of this approach is outlined in Algorithm 3. The proposer is initialized with some random parameter λ_0 and the system is also initialized to some random hypothesis Y_0. At every step of the algorithm, we optimize the proposer $\mathcal{J}(Y_t, \mathcal{X}; \lambda)$ using the CEM algorithm where we use the learned model (i.e., by SampleRank) or the ground truth (for example, accuracy) as the performance metric. The algorithm uses the best proposed hypothesis as the next state, and repeats until it converges to the best encountered hypothesis (hopefully a tight neighborhood around the best hypothesis). Note that the complexity of the computations pertaining to the proposer (computing the within-cluster disparity and between-cluster affinity scores) at each step is at most $O(N_t^2)$ where N_t is the total number of clusters (i.e., the number of $y_i^t \in Y_t$ variables that are "on") at time t. This overcomes the computational barriers of the initial idea that we described in the beginning of this section (where the whole configurations need to be sampled, instantiated and scored).

Remark 1: Recall from Section 2.2 that the CEM optimization requires solving the Equation 2. Our parametrized split-merge proposer (Equation 10) is a linear mixture of two exponential family probability distributions, thus it is a convex function. The summation in Equation 2 is the sum of the logs of convex functions and is therefore concave. Note that maximizing a concave function is equivalent to minimizing its convex counterpart. Thus we are justified in using numeric methods (e.g., L-BFGS) to solve the maximization problem in Equation 2.

5 Experiments

In this section we evaluate the performance of the adaptive proposal distribution that we introduced in the previous section. In the first experiment we compare the adaptive proposer to a traditional uniform split-merge proposer in terms of the number of jumps required to reach within 95% of the ground-truth configuration. Then, in the second experiment we demonstrate the adaptive proposer's ability to identify variables in the configuration space that are incorrectly set. Both sets of experiments evaluate the algorithm on a factor graph used for modeling newswire coreference. Newswire coreference is the problem of clustering textual mentions into real-world entities. For example, a newspaper article about some current event might include textual mentions "Biden", "he", "the Vice President", all of which must be clustered into the same entity (implicitly referring to Vice President Joe Biden).

The factor graph we use for modeling coreference contains a hidden variable for each hypothetical set of mentions. That is, each hidden variable is a binary decision (*yes*, indicating coreferent or *no* otherwise). We can view the graph shown in Figure 1 as a model for coreference where the observed $\mathbf{x}$ variables are the individual mentions (e.g., $\mathbf{x_1}$=Biden, $\mathbf{x_2}$=Obama), the $\mathbf{y}$ variables are binary coreference variables indicating whether a cluster is coreferent, and the factors are the dot products of feature functions and parameters. For a more detailed description of the type of factor graph used for modeling coreference see [4].

This problem is particularly enticing because the factor graph modeling it exhibits combinatorial complexity; in particular, the number of possible instantiations of the graph is Bell(n) number where n is the number of mentions in a particular document. Furthermore, the graph structure precludes an efficient dynamic program since there are no overlapping sub-problems to exploit. Next, we present our initial results based on the following two experiments:

• The purpose of the first experiment is to demonstrate that in the context of SampleRank, the cross entropy based proposer (CEMP) is able to reach the MAP configuration in fewer jumps than the traditional uniform split merge proposer (USMP) that was mentioned in §2. In particular we run SampleRank algorithm with both CEMP and USMP as proposers and record the number of jumps taken when the system gets within 95% of the MAP configuration (according to the ground truth signal).

As previously described, the USMP proposer makes a single jump per local-search step. However, for a fair comparison, we allow the SMP proposer to sample as many jumps as the CEMP proposer would have used to learn its parameters. In particular, CEMP performs a round of learning every five (or ten, see Table 1) jumps. During learning steps, fifty samples are taken for each of the twenty CEM iterations, and the proposer parameters are updated according to the cross entropy method. For non-learning steps, only five samples are taken and the maximum returned as the proposed jump. We similarly allow the uniform distribution to propose 1000 samples every fifth step and five (or ten) samples every other step. As is the case with the CEMP, the maximumal scoring sample is returned as the proposed jump.

We use a subset of the ACE 2004 data-set for newswire coreference. In particular we randomly selected 30 documents, containing between 20 and 115 mentions each. Table 1 compares the number of samples required to get to a configuration within 95% of the true MAP score for both the cross entropy method proposer (CEMP) and the uniform split merge proposer (USMP). The "#samples" column refers to the number of samples between learning the steps.

Proposer	Total # of Samples	Total # of jumps	Average # of jumps per document
SR+CEMP	5	**2942**	**98.1**
SR+USMP	5	3988	132.9
SR+CEMP	10	**2534**	**84.5**
SR+USMP	10	2822	94.1

Table 1: A comparison of the number of samples required to get to a configuration within 95% of the ground-truth using the Cross Entropy Method Proposer (CEMP) and the Uniform Split Merge Proser (USMP) during

The top two rows of Table 1 compare CEMP and USP when learning takes place every five rounds (that is, every five rounds 1000 samples are taken and the maximum returned). We are pleased to see that CEMP requires far fewer jumps than USMP (27% fewer to be exact). The second two rows of the table double the number of samples taken during non-learning rounds to ten. In this case CEMP also requires fewer samples than USMP, but the results are not quite as dramatic. Note that CEMP with only five samples per step does almost as well as USMP with ten samples.

• In the second experiment, we evaluate the cross-entropy proposal distribution's ability to identify incorrect coreference clusters for splitting. For this experiment we took the same thirty ACE documents, initialized each to the ground truth, then randomly selected two clusters to merge into a single incorrect cluster. The strategy of guessing a cluster uniformly at random would result in an accuracy of $1/c$ where c is the number of non-singleton clusters. In fact over the course of 100 samples per document, the uniform split-merge proposer (USMP) was only able to identify the incorrect cluster 15% of the time. However, the cross entropy proposer trained with 10 iterations and 100 samples per iteration was able to achieve a selection accuracy of 50.0%; meaning that half the time CEMP was able to correctly identify the only incorrect cluster in the configuration.

6 Conclusions

In this paper we presented a convergence proof for the SampleRank algorithm. We also developed a decoding algorithm based on CEM that adapts a jump proposer to produce more fruitful jumps using the samples generated by a random walk in the hypothesis space. We presented initial results that evaluates the performance of the new proposer in a real world information extraction domain.

There are a number of open questions and interesting directions for further investigation. The mixing rate of the split and merge (i.e., β in Equation 10) is fixed and not learned. It would be helpful to be able to learn a mixing rate that can generalize across different configuration. The adaptive proposal distribution that we presented uses simple jumps such as traditionally used splits and merges. More complex jumps could be incorporated in the model. These are some of the open problems that we will investigate in future work.

7 Acknowledgments

This work was supported in part by the Center for Intelligent Information Retrieval, in part by Lockheed Martin through prime contract No. FA8650-06-C-7605 from the Air Force Office of Scientific Research, in part by UPenn NSF medium IIS-0803847, in part by The Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grant #IIS-0326249, and by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

References

[1] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. *EMNLP*, 2002.

[2] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[3] Aron Culotta. *Learning and inference in weighted logic with application to natural language processing*. PhD thesis, University of Massachusetts, May 2008.

[4] 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.

[5] P. T. De Boer, D. P. Kroese, and R. Y. Rubinstein. Rare event simulation and combinatorial optimization using cross entropy: estimating buffer overflows in three stages using cross-entropy. In *WSC '02: Proceedings of the 34th conference on Winter simulation*, pages 301–309. Winter Simulation Conference, 2002.

[6] P.T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134, 2004.

[7] N. D. Goodman, V. K. Mansighka, K. Bonawitz D. Roy, and J. B. Tenenbaum. A language for generative models. In *Uncertainty in Artificial Intelligence*, 2008.

[8] Peter Green and Sylvia Richardson. Modeling heterogeneity with and without the dirichlet process. *Scandinavian Journal of Statistics*, 28:355–375, 2001.

[9] III Hal Daumé and Daniel Marcu. Learning as search optimization: approximate large margin methods for structured prediction. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 169–176, New York, NY, USA, 2005. ACM.

[10] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:2002, 2002.

[11] Sonia Jain and Radford M. Neal. A split-merge markov chain monte carlo procedure for the dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13:158–182, 2004.

[12] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[13] Andrew McCallum, Khashayar Rohanimanesh, Michael Wick, Karl Schultz, and Sameer Singh. Factorie: Efficient probabilistic programming via imperative declarations of structure, inference and learning. In *Neural Information Processing Systems(NIPS) Workshop on Probabilistic Programming*, Vancouver, BC, Canda, 2008.

[14] Brian Milch, Bhaskara Marthi, and Stuart Russell. *BLOG: Relational Modeling with Unknown Objects*. PhD thesis, University of California, Berkeley, 2006.

[15] Hanna Pasula and Stuart Russell. Approximate inference for first-order probabilistic languages. In *Proceedings of the International Joint Conferences on Artificial Intelligence 2001 (IJCAI-01)*, 2001.

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