Penalized Expectation Propagation for Graphical Models over Strings[*]

Ryan Cotterell and **Jason Eisner**
Department of Computer Science, Johns Hopkins University
{ryan.cotterell,jason}@cs.jhu.edu

Abstract

We present penalized expectation propagation (PEP), a novel algorithm for approximate inference in graphical models. Expectation propagation is a variant of loopy belief propagation that keeps messages tractable by projecting them back into a given family of functions. Our extension, PEP, uses a structured-sparsity penalty to encourage simple messages, thus balancing speed and accuracy. We specifically show how to instantiate PEP in the case of string-valued random variables, where we adaptively approximate finite-state distributions by variable-order n-gram models. On phonological inference problems, we obtain substantial speedup over previous related algorithms with no significant loss in accuracy.

1 Introduction

Graphical models are well-suited to reasoning about linguistic structure in the presence of uncertainty. Such models typically use discrete random variables, where each variable ranges over a finite set of values such as words or tags. But a variable can also be allowed to range over an *infinite* space of discrete *structures*—in particular, the set of all strings, a case first explored by Bouchard-Côté et al. (2007).

This setting arises because human languages make use of many word forms. These strings are systematically related in their spellings due to linguistic processes such as morphology, phonology, abbreviation, copying error and historical change. To analyze or predict novel strings, we can model the joint distribution of many related strings at once. Under a graphical model, the joint probability of an assignment tuple is modeled as a product of potentials on sub-tuples, each of which is usually modeled in turn by a weighted finite-state machine.

In general, we wish to infer the values of unknown strings in the graphical model. Deterministic approaches to this problem have focused on belief propagation (BP), a message-passing algorithm that is exact on acyclic graphical models and approximate on cyclic ("loopy") ones (Murphy et al., 1999). But in both cases, further heuristic approximations of the BP messages are generally used for speed.

In this paper, we develop a more principled and flexible way to approximate the messages, using variable-order n-gram models.

We first develop a version of expectation propagation (EP) for string-valued variables. EP offers a principled way to approximate BP messages by distributions from a *fixed* family—e.g., by trigram models. Each message update is found by minimizing a certain KL-divergence (Minka, 2001a).

Second, we generalize to variable-order models. To do this, we augment EP's minimization problem with a novel penalty term that keeps the number of n-grams finite. In general, we advocate penalizing more "complex" messages (in our setting, large finite-state acceptors). Complex messages are slower to construct, and slower to use in later steps.

Our penalty term is formally similar to regularizers that encourage structured sparsity (Bach et al., 2011; Martins et al., 2011). Like a regularizer, it lets us use a more expressive family of distributions, secure in the knowledge that we will use only as many of the parameters as we really need for a "pretty good" fit. But *why* avoid using more parameters? Regularization seeks better *generalization* by not overfitting the model to the data. By contrast, we already have a model and are merely doing inference. We seek better *runtime* by not over-fussing about capturing the model's marginal distributions.

Our "penalized EP" (PEP) inference strategy is applicable to any graphical model with complex messages. In this paper, we focus on strings, and show how PEP speeds up inference on the computational phonology model of Cotterell et al. (2015).

We provide further details, tutorial material, and results in the appendices (supplementary material).

[*]This material is based upon work supported by the National Science Foundation under Grant No. 1423276, and by a Fulbright Research Scholarship to the first author.

2 Background

Graphical models over strings are in fairly broad use. Linear-chain graphical models are equivalent to cascades of finite-state transducers, which have long been used to model stepwise derivational processes such as speech production (Pereira and Riley, 1997) and transliteration (Knight and Graehl, 1998). Tree-shaped graphical models have been used to model the evolution and speciation of word forms, in order to reconstruct ancient languages (Bouchard-Côté et al., 2007; Bouchard-Côté et al., 2008) and discover cognates in related languages (Hall and Klein, 2010; Hall and Klein, 2011). Cyclic graphical models have been used to model morphological paradigms (Dreyer and Eisner, 2009; Dreyer and Eisner, 2011) and to reconstruct phonological underlying forms (Cotterell et al., 2015). All of these graphical models, except Dreyer's, happen to be *directed* ones. And all of these papers, except Bouchard-Côté's, use *deterministic* inference methods—based on BP.

2.1 Graphical models over strings

A directed or undirected graphical model describes a joint probability distribution over a set of random variables. To perform inference *given* a setting of the model parameters *and* observations of some variables, it is convenient to construct a *factor graph* (Kschischang et al., 2001). A factor graph is a finite bipartite graph whose vertices are the random variables $\{V_1, V_2, \ldots\}$ and the factors $\{F_1, F_2, \ldots\}$. Each factor F is a function of the variables that it is connected to; it returns a non-negative real number that depends on the values of those variables. We define our factor graph so that the posterior probability $p(V_1 = v_1, V_2 = v_2, \ldots \mid \text{observations})$, as defined by the original graphical model, can be computed as proportional to the product of the numbers returned by all the factors when $V_1 = v_1, V_2 = v_2, \ldots$.

In a graphical model *over strings*, each random variable V is permitted to range over the strings Σ^* where Σ is a fixed alphabet. As in previous work, we will assume that each factor F connected to d variables is a d-way rational relation, i.e., a function that can be computed by a d-tape weighted finite-state acceptor (Elgot and Mezei, 1965; Mohri et al., 2002; Kempe et al., 2004). The weights fall in the semiring $(\mathbb{R}, +, \times)$: F's return value is the *total* weight of all paths that accept the d-tuple of strings, where a path's weight is the *product* of its arcs' weights. So our model marginalizes over possible paths in F.

2.2 Inference by (loopy) belief propagation

Inference seeks the posterior marginal probabilities $p(V_i = v \mid \text{observations})$, for each i. BP is an iterative procedure whose "normalized beliefs" converge to exactly these marginals if the factor graph is acyclic (Pearl, 1988). In the cyclic case, the normalized beliefs still typically converge and can be used as approximate marginals (Murphy et al., 1999).

A full presentation of BP for graphical models over strings can be found in Dreyer and Eisner (2009). We largely follow their notation. $\mathcal{N}(X)$ represents the set of neighbors of X in the factor graph.

For each edge in the factor graph, between a factor F and a variable V, BP maintains two *messages*, $\mu_{V\to F}$ and $\mu_{F\to V}$. Each of these is a function over the possible values v of variable V, mapping each v to a non-negative score. BP also maintains another such function, the *belief* b_V, for each variable V.

In general, each message or belief should be regarded as giving only *relative* scores for the different v. Rescaling it by a positive constant would only result in rescaling other messages and beliefs, which would not change the final normalized beliefs. The *normalized belief* is the probability distribution $\hat{b}_V$ such that each $\hat{b}_V(v)$ is proportional to $b_V(v)$.

The basic BP algorithm is just to repeatedly select and update a function until convergence. The rules for updating $\mu_{V\to F}$, $\mu_{F\to V}$, and b_V, given the set of "neighboring" messages in each case, can be found as equations (2)–(4) of Dreyer and Eisner (2009). (We will give the EP variants in section 4.)

Importantly, that paper shows that for graphical models over strings, each BP update can be implemented via standard finite-state operations of composition, projection, and intersection. Each message or belief is represented as a weighted finite-state acceptor (WFSA) that scores all strings $v \in \Sigma^*$.

2.3 The need for approximation

BP is generally only used directly for short cascades of finite-state transducers (Pereira and Riley, 1997; Knight and Graehl, 1998). Alas, in other graphical models over strings, the BP messages—which are acceptors—become too large to be practical.

In cyclic factor graphs, where exact inference for strings can be *undecidable*, the WFSAs can become *unboundedly* large as they are iteratively updated around a cycle (Dreyer and Eisner, 2009). Even in an acyclic graph (where BP is exact), the finite-state operations quickly lead to large WFSAs. Each intersection or composition is a Cartesian product construction, whose output's size (number of automaton states) may be as large as the *product* of its inputs' sizes. Combining many of these operations leads to exponential blowup.

3 Variational Approximation of WFSAs

To address this difficulty through EP (section 4), we will need the ability to approximate any probability distribution p that is given by a WFSA, by choosing a "simple" distribution from a family $\mathcal{Q}$.

Take $\mathcal{Q}$ to be a family of log-linear distributions

$$q_{\boldsymbol{\theta}}(v) \overset{\text{def}}{=} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(v)) \,/\, Z_{\boldsymbol{\theta}} \qquad (\forall v \in \Sigma^*) \tag{1}$$

where $\boldsymbol{\theta}$ is a weight vector, $\boldsymbol{f}(v)$ is a feature vector that describes v, and $Z_{\boldsymbol{\theta}} \overset{\text{def}}{=} \sum_{v \in \Sigma^*} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(v))$ so that $\sum_v q_{\boldsymbol{\theta}}(v) = 1$. Notice that the featurization function $\boldsymbol{f}$ specifies the family $\mathcal{Q}$, while the variational parameters $\boldsymbol{\theta}$ specify a particular $q \in \mathcal{Q}$.[1]

We project p into $\mathcal{Q}$ via inclusive KL divergence:

$$\boldsymbol{\theta} = \operatorname{argmin}_{\boldsymbol{\theta}} \mathrm{D}(p \,||\, q_{\boldsymbol{\theta}}) \tag{2}$$

Now $q_{\boldsymbol{\theta}}$ approximates p, and has support everywhere that p does. We can get finer-grained approximations by expanding $\boldsymbol{f}$ to extract more features: however, $\boldsymbol{\theta}$ is then larger to store and slower to find.

3.1 Finding $\boldsymbol{\theta}$

Solving (2) reduces to maximizing $-H(p, q_{\boldsymbol{\theta}}) = \mathbb{E}_{v \sim p}[\log q_{\boldsymbol{\theta}}(v)]$, the log-likelihood of $q_{\boldsymbol{\theta}}$ on an "infinite sample" from p. This is similar to fitting a log-linear model to data (without any regularization: we want $q_{\boldsymbol{\theta}}$ to fit p as well as possible). This objective is concave and can be maximized by following its gradient $\mathbb{E}_{v \sim p}[\boldsymbol{f}(v)] - \mathbb{E}_{v \sim q_{\boldsymbol{\theta}}}[\boldsymbol{f}(v)]$. Often it is also possible to optimize $\boldsymbol{\theta}$ in closed form, as we will see later. Either way, the optimal $q_{\boldsymbol{\theta}}$ matches p's *expected* feature vector: $\mathbb{E}_{v \sim q_{\boldsymbol{\theta}}}[\boldsymbol{f}(v)] = \mathbb{E}_{v \sim p}[\boldsymbol{f}(v)]$. This inspired the name "expectation propagation."

3.2 Working with $\boldsymbol{\theta}$

Although p is defined by an arbitrary WFSA, we can represent $q_{\boldsymbol{\theta}}$ quite simply by just storing the parameter vector $\boldsymbol{\theta}$. We will later take sums of such vectors to construct product distributions: observe that under (1), $q_{\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2}(v)$ is proportional to $q_{\boldsymbol{\theta}_1}(v) \cdot q_{\boldsymbol{\theta}_2}(v)$.

We will also need to construct WFSA versions of these distributions $q_{\boldsymbol{\theta}} \in \mathcal{Q}$, and of other log-linear functions (messages) that may not be normalizable into distributions. Let $\text{ENCODE}(\boldsymbol{\theta})$ denote a WFSA that accepts each $v \in \Sigma^*$ with weight $\exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(v))$.

3.3 Substring features

To obtain our family $\mathcal{Q}$, we must design $\boldsymbol{f}$. Our strategy is to choose a set of "interesting" substrings $\mathcal{W}$. For each $w \in \mathcal{W}$, define a feature function "How many times does w appear as a substring of v?" Thus, $\boldsymbol{f}(v)$ is simply a vector of counts (nonnegative integers), indexed by the substrings in $\mathcal{W}$.

A natural choice of $\mathcal{W}$ is the set of all n-grams for fixed n. In this case, $\mathcal{Q}$ turns out to be equivalent to the family of n-gram language models.[2] Already in previous work ("variational decoding"), we used (2) with this family to approximate WFSAs or weighted hypergraphs that arose at runtime (Li et al., 2009).

Yet a fixed n is not ideal. If $\mathcal{W}$ is the set of bigrams, one might do well to add the trigram `the`—perhaps because `the` is "really" a bigram (counting the digraph `th` as a single consonant), or because the bigram model fails to capture how common `the` is under p. Adding `the` to $\mathcal{W}$ ensures that q_θ will now match p's expected count for this trigram. Doing this should not require adding all $|\Sigma|^3$ trigrams.

By including strings of mixed lengths in $\mathcal{W}$ we get *variable-order* Markov models (Ron et al., 1996).

3.4 Arbitrary FSA-based features

More generally, let $\mathcal{A}$ be any *unambiguous and complete* finite-state acceptor: that is, any $v \in \Sigma^*$ has exactly one accepting path in $\mathcal{A}$. For each arc or final state a in $\mathcal{A}$, we can define a feature function "How

[1]To be precise, we take $\mathcal{Q} = \{q_{\boldsymbol{\theta}} : Z_{\boldsymbol{\theta}} \text{ is finite}\}$. For example, $\boldsymbol{\theta} = \mathbf{0}$ is excluded because then $Z_{\boldsymbol{\theta}} = \sum_{v \in \Sigma^*} \exp 0 = \infty$. Aside from this restriction, $\boldsymbol{\theta}$ may be any vector over $\mathbb{R} \cup \{-\infty\}$. We allow $-\infty$ since it is a feature's optimal weight if $p(v) = 0$ for all v with that feature: then $q_{\boldsymbol{\theta}}(v) = 0$ for such strings as well. (Provided that $\boldsymbol{f}(v) \geq \mathbf{0}$, as we will ensure.)

[2]Provided that we include special n-grams that match at the boundaries of v. See Appendix B.2 for details.

many times is a used when $\mathcal{A}$ accepts v?” Thus, $\boldsymbol{f}(v)$ is again a vector of non-negative counts.

Section 6 gives algorithms for this general setting. We implement the previous section as a special case, constructing $\mathcal{A}$ so that its arcs essentially correspond to the substrings in $\mathcal{W}$. This encodes a variable-order Markov model as an FSA similarly to (Allauzen et al., 2003); see Appendix B.4 for details.

In this general setting, ENCODE$(\boldsymbol{\theta})$ just returns a *weighted* version of $\mathcal{A}$ where each arc or final state a has weight $\exp \theta_a$ in the $(+, \times)$ semiring. Thus, this WFSA accepts each v with weight $\exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(v))$.

3.5 Adaptive featurization

How do we choose $\mathcal{W}$ (or $\mathcal{A}$)? Expanding $\mathcal{W}$ will allow better approximations to p—but at greater computational cost. We would like $\mathcal{W}$ to include just the substrings needed to approximate a *given* p well. For instance, if p is concentrated on a few high-probability strings, then a good $\mathcal{W}$ might contain those full strings (with positive weights), plus some shorter substrings that help model the rest of p.

To select $\mathcal{W}$ at runtime in a way that adapts to p, let us say that $\boldsymbol{\theta}$ is actually an infinite vector with weights for *all possible* substrings, and define $\mathcal{W} = \{w \in \Sigma^* : \theta_w \neq 0\}$. Provided that $\mathcal{W}$ stays finite, we can store $\boldsymbol{\theta}$ as a map from substrings to *nonzero* weights. We keep $\mathcal{W}$ small by replacing (2) with

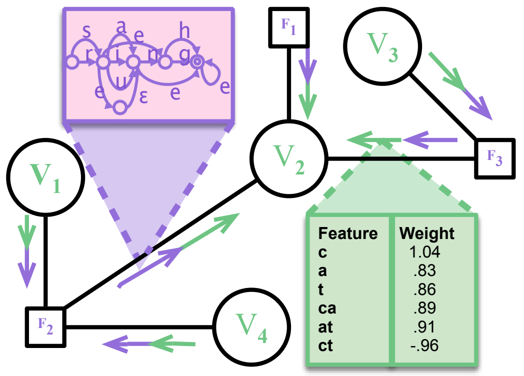
$$\boldsymbol{\theta} = \operatorname{argmin}_{\boldsymbol{\theta}} \mathrm{D}(p \,||\, q_{\boldsymbol{\theta}}) + \lambda \cdot \Omega(\boldsymbol{\theta}) \qquad (3)$$

where $\Omega(\boldsymbol{\theta})$ measures the complexity of this $\mathcal{W}$ or the corresponding $\mathcal{A}$. Small WFSAs ensure fast finite-state operations, so ideally, $\Omega(\boldsymbol{\theta})$ should measure the size of ENCODE$(\boldsymbol{\theta})$. Choosing $\lambda > 0$ to be large will then emphasize speed over accuracy.

Section 6.1 will extend section 6’s algorithms to approximately minimize the new objective (3). Formally this objective resembles regularized log-likelihood. However, $\Omega(\boldsymbol{\theta})$ is not a regularizer—as section 1 noted, we have no statistical reason to avoid “overfitting” $\hat{p}$, only a computational one.

4 Expectation Propagation

Recall from section 2.2 that for each variable V, the BP algorithm maintains several nonnegative functions that score V’s possible values v: the messages $\mu_{V \to F}$ and $\mu_{F \to V}$ ($\forall F \in \mathcal{N}(V)$), and the belief b_V.



Feature	Weight
c	1.04
a	.83
t	.86
ca	.89
at	.91
ct	-.96

Figure 1: Information flowing toward V_2 in EP (reverse flow not shown). The factors work with purple μ messages represented by WFSAs, while the variables work with green $\boldsymbol{\theta}$ messages represented by log-linear weight vectors. The green table shows a $\boldsymbol{\theta}$ message: a sparse weight vector that puts high weight on the string `cat`.

EP is a variant in which all of these are forced to be log-linear functions from the same family, namely $\exp(\boldsymbol{\theta} \cdot \boldsymbol{f}_V(v))$. Here $\boldsymbol{f}_V$ is the featurization function we’ve chosen for variable V.[3] We can represent these functions by their parameter vectors—let us call those $\boldsymbol{\theta}_{V \to F}$, $\boldsymbol{\theta}_{F \to V}$, and $\boldsymbol{\theta}_V$ respectively.

4.1 Passing messages through variables

What happens to BP’s update equations in this setting? According to BP, the belief b_V is the pointwise product of all “incoming” messages to V. But as we saw in section 3.2, pointwise products are *far easier* in EP’s restricted setting! Instead of intersecting several WFSAs, we can simply add several vectors:

$$\boldsymbol{\theta}_V = \sum_{F' \in \mathcal{N}(V)} \boldsymbol{\theta}_{F' \to V} \qquad (4)$$

Similarly, the “outgoing” message from V to factor F is the pointwise product of all “incoming” messages *except* the one from F. This message $\boldsymbol{\theta}_{V \to F}$ can be computed as $\boldsymbol{\theta}_V - \boldsymbol{\theta}_{F \to V}$, which adjusts (4).[4] We never store this but just compute it on demand.

[3]A single graphical model might mix categorical variables, continuous variables, orthographic strings over (say) the Roman alphabet, and phonological strings over the International Phonetic Alphabet. These different data types certainly require different featurization functions. Moreover, even when two variables have the same type, we could choose to approximate their marginals differently, e.g., with bigram vs. trigram features.

[4]If features can have $-\infty$ weight (footnote 1), this trick might need to subtract $-\infty$ from $-\infty$ (the log-space version of $0/0$). That gives an undefined result, but it turns out that any result will do—it makes no difference to the subsequent beliefs.

4.2 Passing messages through factors

Our factors are weighted finite-state machines, so their messages still require finite-state computations, as shown by the purple material in Figure 1. These computations are just as in BP. Concretely, let F be a factor of degree d, given as a d-tape machine. We can compute a belief *at this factor* by joining F with d WFSAs that represent its d incoming messages, namely $\textsc{Encode}(\boldsymbol{\theta}_{V' \to F})$ for $V' \in \mathcal{N}(F)$. This gives a new d-tape machine, b_F. We then obtain each outgoing message $\mu_{F \to V}$ by projecting b_F onto its V tape, but removing (dividing out) the weights that were contributed (multiplied in) by $\boldsymbol{\theta}_{V \to F}$.[5]

4.3 Getting from factors back to variables

Finally, we reach the only tricky step. Each resulting $\mu_{F \to V}$ is a possibly large WFSA, so we must *force it back into our log-linear family* to get an updated approximation $\boldsymbol{\theta}_{F \to V}$. One cannot directly employ the methods of section 3, because KL divergence is only defined between probability distributions. ($\mu_{F \to V}$ might not be normalizable into a distribution, nor is its best approximation necessarily normalizable.)

The EP trick is to use section 3 to instead approximate the belief at V, which *is* a distribution, and *then* reconstruct the approximate message to V that would have produced this approximated belief. The "unapproximated belief" $\hat{p}_V$ resembles (4): it multiplies the unapproximated message $\mu_{F \to V}$ by the current values of all *other* messages $\boldsymbol{\theta}_{F' \to V}$. We know the product of those *other* messages, $\boldsymbol{\theta}_{V \to F}$, so

$$\hat{p}_V := \mu_{F \to V} \odot \mu_{V \to F} \tag{5}$$

where the pointwise product $\odot$ is carried out by WFSA intersection and $\mu_{V \to F} \overset{\text{def}}{=} \textsc{Encode}(\boldsymbol{\theta}_{V \to F})$.

We now apply section 3 to choose $\boldsymbol{\theta}_V$ such that $q_{\boldsymbol{\theta}_V}$ is a good approximation of the WFSA $\hat{p}_V$. Finally, to preserve (4) as an invariant, we reconstruct

$$\boldsymbol{\theta}_{F \to V} := \boldsymbol{\theta}_V - \boldsymbol{\theta}_{V \to F} \tag{6}$$

[5]This is equivalent to computing each $\mu_{F \to V}$ by "generalized composition" of F with the $d-1$ messages to F from its *other* neighbors V'. The operations of join and generalized composition were defined by Kempe et al. (2004).

In the simple case $d = 2$, F is just a weighted finite-state transducer mapping V' to V, and computing $\mu_{F \to V}$ reduces to composing $\textsc{Encode}(\boldsymbol{\theta}_{V' \to F})$ with F and projecting the result onto the output tape. In fact, one can assume WLOG that $d \leq 2$, enabling the use of popular finite-state toolkits that handle at most 2-tape machines. See Appendix B.10 for the construction.

In short, EP combines $\mu_{F \to V}$ with $\boldsymbol{\theta}_{V \to F}$, then approximates the result $\hat{p}_V$ by $\boldsymbol{\theta}_V$ before removing $\boldsymbol{\theta}_{V \to F}$ again. Thus EP is approximating $\mu_{F \to V}$ by

$$\boldsymbol{\theta}_{F \to V} := \operatorname*{argmin}_{\boldsymbol{\theta}} \mathrm{D}(\underbrace{\mu_{F \to V} \odot \mu_{V \to F}}_{=\hat{p}_V} \,||\, \underbrace{q_{\boldsymbol{\theta}} \odot \mu_{V \to F}}_{=\boldsymbol{\theta}_V}) \tag{7}$$

in a way that updates not only $\boldsymbol{\theta}_{F \to V}$ but also $\boldsymbol{\theta}_V$.

Wisely, this objective focuses on approximating the message's scores for the *plausible* values v. Some values v may have $\hat{p}_V(v) \approx 0$, perhaps because *another* incoming message $\boldsymbol{\theta}_{F' \to V}$ rules them out. It does not much harm the objective (7) if these $\mu_{F \to V}(v)$ are poorly approximated by $q_{\boldsymbol{\theta}_{F \to V}}(v)$, since the overall belief is still roughly correct.

Our *penalized* EP simply adds $\lambda \cdot \Omega(\boldsymbol{\theta})$ into (7).

4.4 The EP algorithm: Putting it all together

To run EP (or PEP), initialize all $\boldsymbol{\theta}_V$ and $\boldsymbol{\theta}_{F \to V}$ to $\mathbf{0}$, and then loop repeatedly over the nodes of the factor graph. When visiting a factor F, $\textsc{Encode}$ its incoming messages $\boldsymbol{\theta}_{V \to F}$ (computed on demand) as WFSAs, construct a belief b_F, and update the outgoing WFSA messages $\mu_{F \to V}$. When visiting a variable V, iterate $K \geq 1$ times over its incoming WFSA messages: for each incoming $\mu_{F \to V}$, compute the unapproximated belief $\hat{p}_V$ via (5), then update $\boldsymbol{\theta}_V$ to approximate $\hat{p}_V$, then update $\boldsymbol{\theta}_{F \to V}$ via (6).

For possibly faster convergence, one can alternate "forward" and "backward" sweeps. Visit the factor graph's nodes in a fixed order (given by an approximate topological sort). At a factor, update the outgoing WFSA messages to *later* variables only. At a variable, approximate only those incoming WFSA messages from *earlier* factors (all the outgoing messages $\boldsymbol{\theta}_{V \to F}$ will be recomputed on demand). Note that both cases examine all incoming messages. After each sweep, reverse the node ordering and repeat.

If gradient ascent is used to find the $\boldsymbol{\theta}_V$ that approximates $\hat{p}_V$, it is wasteful to optimize to convergence. After all, the optimization problem will keep changing as the messages change. Our implementation improves $\boldsymbol{\theta}_V$ by only a single gradient step on each visit to V, since V will be visited repeatedly.

See Appendix A for an alternative view of EP.

5 Related Approximation Methods

We have presented EP as a method for simplifying a variable's incoming messages during BP. The variable's outgoing messages are pointwise products of the incoming ones, so they become simple too. Past work has used approximations with a similar flavor.

Hall and Klein (2011) heuristically predetermine a short, fixed list of plausible values for V that were observed elsewhere in their dataset. This list is analogous to our $\boldsymbol{\theta}_V$. After updating $\mu_{F \to V}$, they force $\mu_{F \to V}(v)$ to 0 for all v outside the list, yielding a *finite* message that is analogous to our $\boldsymbol{\theta}_{F \to V}$.

Our own past papers are similar, except they *adaptively* set the "plausible values" list to $\bigcup_{F' \in \mathcal{N}(V)} k\text{-BEST}(\mu_{F' \to V})$. These strings are favored by at least one of the *current* messages to V (Dreyer and Eisner, 2009; Dreyer and Eisner, 2011; Cotterell et al., 2015). Thus, simplifying one of V's incoming messages considers all of them, as in EP.

The above methods *prune* each message, so may prune correct values. Hall and Klein (2010) avoid this: they fit a full bigram model by inclusive KL divergence, which refuses to prune *any* values (see section 3). Specifically, they minimized $\mathrm{D}(\mu_{F \to V} \odot \tau \mid\mid q_{\boldsymbol{\theta}} \odot \tau)$, where τ was a simple fixed function (a 0-gram model) included so that they were working with distributions (see section 4.3). This is very similar to our (7). Indeed, Hall and Klein (2010) found their procedure "reminiscent of EP," hinting that τ was a surrogate for a real $\mu_{V \to F}$ term. Dreyer and Eisner (2009) had also suggested EP as future work.

EP has been applied only twice before in the NLP community. Daumé III and Marcu (2006) used EP for query summarization (following Minka and Lafferty (2003)'s application to an LDA model with fixed topics) and Hall and Klein (2012) used EP for rich parsing. However, these papers inferred a single structured variable connected to all factors (as in the traditional presentation of EP—see Appendix A), rather than inferring many structured variables connected in a sparse graphical model.

We regard EP as a generalization of loopy BP for just this setting: graphical models with large or unbounded variable domains. Of course, we are not the first to use such a scheme; e.g., Qi (2005, chapter 2) applies EP to linear-chain models with both continuous and discrete hidden states. We believe that EP should also be broadly useful in NLP, since it naturally handles joint distributions over the kinds of structured variables that arise in NLP.

6 Two Methods for Optimizing $\boldsymbol{\theta}$

We now fill in details. If the feature set is defined by an unambiguous FSA $\mathcal{A}$ (section 3.4), two methods exist to max $\mathbb{E}_{v \sim p}[\log q_{\boldsymbol{\theta}}(v)]$ as section 3.1 requires.

Closed-form. Determine how often $\mathcal{A}$ would traverse each of its arcs, in expectation, when reading a random string drawn from p. We would obtain an optimal $\textsc{Encode}(\boldsymbol{\theta})$ by, at each state of $\mathcal{A}$, setting the weights of the arcs from that state to be proportional to these counts while summing to 1.[6] Thus, the logs of these arc weights give an optimal $\boldsymbol{\theta}$.

For example, in a trigram model, the probability of the `c` arc from the `ab` state is the expected count of `abc` (according to p) divided by the expected count of `ab`. Such expected substring counts can be found by the method of Allauzen et al. (2003). For general $\mathcal{A}$, we can use the method sketched by Li et al. (2009, footnote 9): intersect the WFSA for p with the unweighted FSA $\mathcal{A}$, and then run the forward-backward algorithm to determine the posterior count of each arc in the result. This tells us the expected total number of traversals of each arc in $\mathcal{A}$, if we have kept track of which arcs in the intersection of p with $\mathcal{A}$ were derived from which arcs in $\mathcal{A}$. That bookkeeping can be handled with an expectation semiring (Eisner, 2002), or simply with backpointers.

Gradient ascent. For any given $\boldsymbol{\theta}$, we can use the WFSAs p and $\textsc{Encode}(\boldsymbol{\theta})$ to exactly compute $\mathbb{E}_{v \sim p}[\log q_{\boldsymbol{\theta}}(v)] = -H(p, q_{\boldsymbol{\theta}})$ (Cortes et al., 2006). We can tune $\boldsymbol{\theta}$ to globally maximize this objective.

The technique is to intersect p with $\textsc{Encode}(\boldsymbol{\theta})$, after lifting their weights into the expectation semiring via the mappings $k \mapsto \langle k, 0 \rangle$ and $k \mapsto \langle 0, \log k \rangle$ respectively. Summing over all paths of this intersection via the forward algorithm yields $\langle Z, r \rangle$ where Z is the normalizing constant for p. We also sum over paths of $\textsc{Encode}(\boldsymbol{\theta})$ to get the normalizing constant $Z_{\boldsymbol{\theta}}$. Now the desired objective is $r/Z - \log Z_{\boldsymbol{\theta}}$. Its *gradient* with respect to $\boldsymbol{\theta}$ can be found by back-propagation, or equivalently by the forward-backward algorithm (Li and Eisner, 2009).

An overlarge gradient step can leave the feasible space (footnote 1) by driving $Z_{\boldsymbol{\theta}_V}$ to ∞ and thus driving (2) to ∞ (Dreyer, 2011, section 2.8.2). In this case, we try again with reduced stepsize.

[6]This method always yields a probabilistic FSA, i.e., the arc weights are locally normalized probabilities. This does not sacrifice any expressiveness; see Appendix B.7 for discussion.

6.1 Optimizing θ with a penalty

Now consider the penalized objective (3). Ideally, $\Omega(\boldsymbol{\theta})$ would count the number of nonzero weights in $\boldsymbol{\theta}$—or better, the number of arcs in $\textsc{Encode}(\boldsymbol{\theta})$. But it is not known how to efficiently minimize the resulting discontinuous function. We give two approximate methods, based on the two methods above.

Proximal gradient. Leaning on recent advances in sparse estimation, we replace this $\Omega(\boldsymbol{\theta})$ with a convex surrogate whose partial derivative with respect to each θ_w is undefined at $\theta_w = 0$ (Bach et al., 2011). Such a penalty tends to create sparse optima.

A popular surrogate is an ℓ_1 penalty, $\Omega(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \sum_w |\boldsymbol{\theta}_w|$. However, ℓ_1 would not recognize that $\boldsymbol{\theta}$ is simpler with the features $\{\texttt{ab}, \texttt{abc}, \texttt{abd}\}$ than with the features $\{\texttt{ab}, \texttt{pqr}, \texttt{xyz}\}$. The former leads to a smaller WFSA encoding. In other words, it is cheaper to add abd once abc is already present, as a state already exists that represents the context ab.

We would thus like the penalty to be the number of distinct *prefixes* in the set of nonzero features,

$$|\{u \in \Sigma^* : (\exists x \in \Sigma^*)\, \theta_{ux} \neq 0\}|, \tag{8}$$

as this is the number of ordinary arcs in $\textsc{Encode}(\boldsymbol{\theta})$ (see Appendix B.4). Its convex surrogate is

$$\Omega(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \sum_{u \in \Sigma^*} \sqrt{\sum_{x \in \Sigma^*} \theta_{ux}^2} \tag{9}$$

This *tree-structured group lasso* (Nelakanti et al., 2013) is an instance of group lasso (Yuan and Lin, 2006) where the string $w = \texttt{abd}$ belongs to four groups, corresponding to its prefixes $u = \epsilon, u = \texttt{a}, u = \texttt{ab}, u = \texttt{abd}$. Under group lasso, moving θ_w away from 0 increases $\Omega(\boldsymbol{\theta})$ by $\lambda|\theta_w|$ (just as in ℓ_1) *for each group in which w is the only nonzero feature*. This penalizes for the new WFSA arcs needed for these groups. There are also increases due to w's other groups, but these are smaller, especially for groups with many strongly weighted features.

Our objective (3) is now the sum of a differentiable convex function (2) and a *particular* non-differentiable convex function (9). We minimize it by proximal gradient (Parikh and Boyd, 2013). At each step, this algorithm first takes a gradient step as in section 6 to improve the differentiable term, and then applies a "proximal operator" to jump to

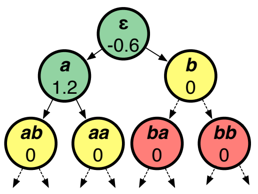


Figure 2: Active set method, showing the infinite tree of all features for the alphabet $\Sigma = \{a, b\}$. The green nodes currently have non-zero weights. The yellow nodes are on the frontier and are *allowed* to become non-zero, but the penalty function is still keeping them at 0. The red nodes are not yet considered, forcing them to remain at 0.

a nearby point that improves the non-differentiable term. The proximal operator for tree-structured group lasso (9) can be implemented with an efficient recursive procedure (Jenatton et al., 2011).

What if $\boldsymbol{\theta}$ is ∞-dimensional because we allow all n-grams as features? Paul and Eisner (2012) used just this feature set in a dual decomposition algorithm. Like them, we rely on an *active set* method (Schmidt and Murphy, 2010). We fix abcd's weight at 0 until abc's weight becomes nonzero (if ever);[7] only then does feature abc become "active." Thus, at a given step, we only have to compute the gradient with respect to the currently nonzero features (green nodes in Figure 2) and their immediate children (yellow nodes). This hierarchical inclusion technique ensures that we only consider a small, finite subset of all n-grams at any given iteration of optimization.

Closed-form with greedy growing. There are existing methods for estimating variable-order n-gram language models from data, based on either "shrinking" a high-order model (Stolcke, 1998) or "growing" a low-order one (Siivola et al., 2007).

We have designed a simple "growing" algorithm to estimate such a model from a WFSA p. It approximately minimizes the objective (3) where $\Omega(\boldsymbol{\theta})$ is given by (8). We enumerate all n-grams $w \in \Sigma^*$ in decreasing order of expected count (this can be done efficiently using a priority queue). We add w to $\mathcal{W}$ if we *estimate* that it will decrease the objective. Every so often, we measure the *actual* objective (just as in the gradient-based methods), and we stop if it is no longer improving. Algorithmic details are given in Appendices B.8–B.9.

[7]Paul and Eisner (2012) also required bcd to have nonzero weight, observing that abcd is a *conjunction* $\texttt{abc} \wedge \texttt{bcd}$ (McCallum, 2003). This added test would be wise for us too.

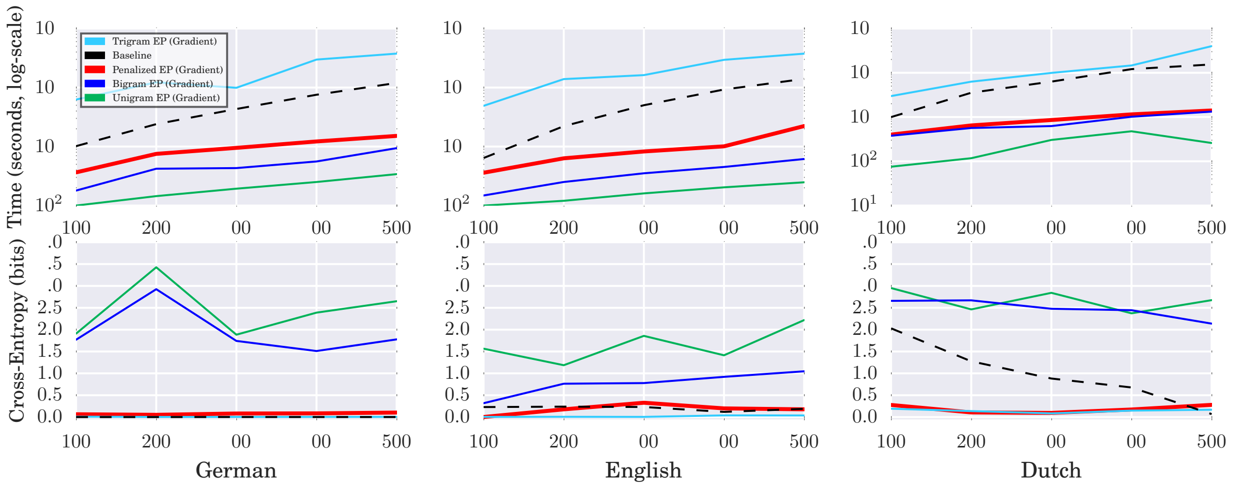


Figure 3: Inference on 15 factor graphs (3 languages × 5 datasets of different sizes). The first row shows the total runtime (logscale) of each inference method. The second row shows the accuracy, as measured by the negated log-probability that the inferred belief at a variable assigns to its gold-standard value, averaged over "underlying morpheme" variables. At this penalty level ($\lambda = 0.01$), PEP [thick line] is faster than the pruning baseline of Cotterell et al. (2015) [dashed line] and much faster than trigram EP, yet is about as accurate. (For Dutch with sparse observations, it is considerably more accurate than baseline.) Indeed, PEP is nearly as fast as bigram EP, which has terrible accuracy. An ideal implementation of PEP would be faster yet (see Appendix B.5). Further graphs are in Appendix C.

7 Experiments and Results

Our experimental design aims to answer three questions. (1) Is our algorithm able to beat a strong baseline (adaptive pruning) in a non-trivial model? (2) Is PEP actually better than ordinary EP, given that the structured sparsity penalty makes it more algorithmically complex? (3) Does the λ parameter successfully trade off between speed and accuracy?

All experiments took place using the graphical model over strings for the discovery of underlying phonological forms introduced in Cotterell et al. (2015). They write: "Comparing `cats` ([kæts]), `dogs` ([dɔgz]), and `quizzes` ([kwɪzɪz]), we see the English plural morpheme evidently has at least three pronunciations." Cotterell et al. (2015) sought a unifying account of such variation in terms of phonological underlying forms for the morphemes.

In their Bayes net, morpheme underlying forms are latent variables, while word surface forms are observed variables. The factors model underlying-to-surface phonological changes. They learn the factors by Expectation Maximization (EM). Their first E step presents the hardest inference problem because the factors initially contribute no knowledge of the language; so that is the setting we test on here.

Their data are surface phonological forms from the CELEX database (Baayen et al., 1995). For each of 3 languages, we run 5 experiments, by observing the surface forms of 100 to 500 words and running EP to infer the underlying forms of their morphemes. Each of the 15 factor graphs has ≈ 150–700 latent variables, joined by 500–2200 edges to 200–1200 factors of degree 1–3. Variables representing suffixes can have extremely high degree (> 100).

We compare PEP with other approximate inference methods. As our main baseline, we take the approximation scheme actually used by Cotterell et al. (2015), which restricts the domain of a belief to that of the union of 20-best strings of its incoming messages (section 5).We also compare to unpenalized EP with unigram, bigram, and trigram features.

We report both speed and accuracy for all methods. Speed is reported in seconds. Judging accuracy is a bit trickier. The best metric would to be to measure our beliefs' distance from the true marginals or even from the beliefs computed by vanilla loopy BP. Obtaining these quantities, however, would be extremely expensive—even Gibbs sampling is infeasible in our setting, let alone 100-way WFSA intersections. Luckily, Cotterell et al. (2015) provide gold-standard values for the latent variables (underlying forms). Figure 3 shows the negated log-probabilities of these gold strings according to our beliefs, averaged over variables in a given factor graph. Our accuracy is weaker than Cotterell et al. (2015) because we are doing inference with their *initial* (untrained) parameters, a more challenging problem.

Each update to $\boldsymbol{\theta}_V$ consisted of a single step of (proximal) gradient descent: starting at the current value, improve (2) with a gradient step of size $\eta = 0.05$, then (in the adaptive case) apply the proximal operator of (9) with $\lambda = 0.01$. We chose these values by preliminary exploration, taking η small enough to avoid backtracking (section 6.1).

We repeatedly visit variables and factors (section 4.4) in the forward-backward order used by Cotterell et al. (2015). For the first few iterations, when we visit a variable we make $K = 20$ passes over its incoming messages, updating them iteratively to ensure that the high probability strings in the initial approximations are "in the ballpark". For subsequent iterations of message passing we take $K = 1$. For similar reasons, we constrained PEP to use only unigram features on the first iteration, when there are still many viable candidates for each morph.

7.1 Results

The results show that PEP is much faster than the baseline pruning method, as described in Figure 3 and its caption. It mainly achieves better cross-entropy on English and Dutch, and even though it loses on German, it still places almost all of its probability mass on the gold forms. While EP with unigram and bigram approximations are both faster than PEP, their accuracy is poor. Trigram EP is nearly as accurate but even slower than the baseline. The results support the claim that PEP has achieved a "Goldilocks number" of n-grams in its approximation—just enough n-grams to approximate the message well while retaining speed.

Figure 4 shows the effect of λ on the speed-accuracy tradeoff. To compare apples to apples, this experiment fixed the set of $\mu_{F\rightarrow V}$ messages for each variable. Thus, we held the set of beliefs fixed, but measured the size and accuracy of different approximations to these beliefs by varying λ.

These figures show only the results from gradient-based approximation. Closed-form approximation is faster and comparably accurate: see Appendix C.

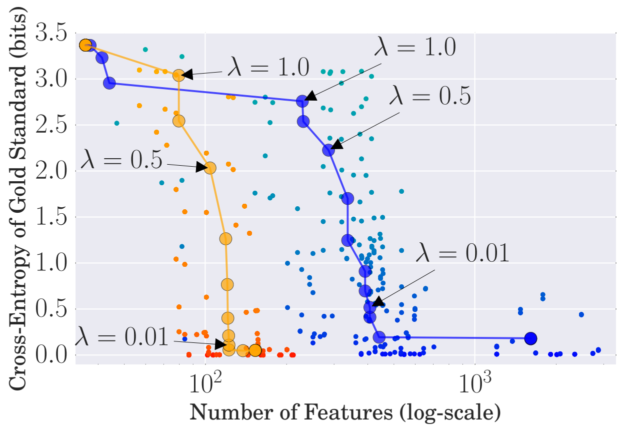


Figure 4: Increasing λ will greatly reduce the number of selected features in a belief—initially without harming accuracy, and then accuracy degrades gracefully. (Number of features has 0.72 correlation with runtime, and is shown on a *log scale* on the x axis.)

Each point shows the result of using PEP to approximate the belief at some latent variable V, using $\mu_{F\rightarrow V}$ messages from running the baseline method on German. Lighter points use larger λ. Orange points are affixes (shorter strings), blue are stems (longer strings). Large circles are averages over all points for a given λ.

8 Conclusion and Future Work

We have presented penalized expectation propagation (PEP), a novel approximate inference algorithm for graphical models, and developed specific techniques for string-valued random variables. Our method integrates structured sparsity directly into inference. Our experiments show large speedups over the strong baseline of Cotterell et al. (2015).

In future, instead of choosing λ, we plan to reduce λ as PEP runs. This serves to gradually refine the approximations, yielding an *anytime algorithm* whose beliefs approach the BP beliefs. Thanks to (7), the coarse messages from early iterations guide the choice of finer-grained messages at later iterations. In this regard, "Anytime PEP" resembles other coarse-to-fine architectures such as generalized A* search (Felzenszwalb and McAllester, 2007).

As NLP turns its attention to lower-resource languages and social media, it is important to model the rich phonological, morphological, and orthographic processes that interrelate words. We hope that the introduction of faster inference algorithms will increase the use of graphical models over strings. We are releasing our code package (see Appendix D).

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing statistical language models. In *Proceedings of ACL*, pages 40–47.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer.

R. Harald Baayen, Richard Piepenbrock, and Leon Gulikers. 1995. The CELEX lexical database on CD-ROM.

Francis Bach, Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, et al. 2011. Convex optimization with sparsity-inducing norms. In S. Sra, S. Nowozin, and S. J. Wright, editors, *Optimization for Machine Learning*. MIT Press.

Alexandre Bouchard-Côté, Percy Liang, Thomas L Griffiths, and Dan Klein. 2007. A probabilistic approach to diachronic phonology. In *Proceedings of EMNLP-CoNLL*, pages 887–896.

Alexandre Bouchard-Côté, Percy Liang, Thomas Griffiths, and Dan Klein. 2008. A probabilistic approach to language change. In *Proceedings of NIPS*.

Victor Chahuneau. 2013. PyFST. `https://github.com/vchahun/pyfst`.

Corinna Cortes, Mehryar Mohri, Ashish Rastogi, and Michael D Riley. 2006. Efficient computation of the relative entropy of probabilistic automata. In *LATIN 2006: Theoretical Informatics*, pages 323–336. Springer.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2014. Stochastic contextual edit distance and probabilistic FSTs. In *Proceedings of ACL*, pages 625–630.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2015. Modeling word forms using latent underlying morphs and phonology. *Transactions of the Association for Computational Linguistics*. To appear.

Hal Daumé III and Daniel Marcu. 2006. Bayesian query-focused summarization. In *Proceedings of ACL*, pages 305–312.

Markus Dreyer and Jason Eisner. 2009. Graphical models over multiple strings. In *Proceedings of EMNLP*, pages 101–110, Singapore, August.

Markus Dreyer and Jason Eisner. 2011. Discovering morphological paradigms from plain text using a Dirichlet process mixture model. In *Proceedings of EMNLP*, pages 616–627, Edinburgh, July.

Markus Dreyer. 2011. *A Non-Parametric Model for the Discovery of Inflectional Paradigms from Plain Text Using Graphical Models over Strings*. Ph.D. thesis, Johns Hopkins University, Baltimore, MD, April.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of ACL*, pages 1–8.

C.C. Elgot and J.E. Mezei. 1965. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68.

Gal Elidan, Ian Mcgraw, and Daphne Koller. 2006. Residual belief propagation: Informed scheduling for asynchronous message passing. In *Proceedings of UAI*.

P. F. Felzenszwalb and D. McAllester. 2007. The generalized A* architecture. *Journal of Artificial Intelligence Research*, 29:153–190.

David Hall and Dan Klein. 2010. Finding cognate groups using phylogenies. In *Proceedings of ACL*.

David Hall and Dan Klein. 2011. Large-scale cognate recovery. In *Proceedings of EMNLP*.

David Hall and Dan Klein. 2012. Training factored PCFGs with expectation propagation. In *Proceedings of EMNLP*.

Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, and Francis Bach. 2011. Proximal methods for hierarchical sparse coding. *The Journal of Machine Learning Research*, 12:2297–2334.

André Kempe, Jean-Marc Champarnaud, and Jason Eisner. 2004. A note on join and auto-intersection of n-ary rational relations. In Loek Cleophas and Bruce Watson, editors, *Proceedings of the Eindhoven FASTAR Days (Computer Science Technical Report 04-40)*, pages 64–78. Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Netherlands, December.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4).

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

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of EMNLP*, pages 40–51.

Zhifei Li, Jason Eisner, and Sanjeev Khudanpur. 2009. Variational decoding for statistical machine translation. In *Proceedings of ACL*, pages 593–601.

André F. T. Martins, Noah A. Smith, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2011. Structured sparsity in structured prediction. In *Proceedings of EMNLP*, pages 1500–1511.

Andrew McCallum. 2003. Efficiently inducing features of conditional random fields. In *Proceedings of UAI*.

Thomas Minka and John Lafferty. 2003. Expectation-propagation for the generative aspect model. In *Proceedings of UAI*.

Thomas P Minka. 2001a. Expectation propagation for approximate Bayesian inference. In *Proceedings of UAI*, pages 362–369.

Thomas P. Minka. 2001b. *A Family of Algorithms for Approximate Bayesian Inference*. Ph.D. thesis, Massachusetts Institute of Technology, January.

Thomas Minka. 2005. Divergence measures and message passing. Technical Report MSR-TR-2005-173, Microsoft Research, January.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Mehryar Mohri. 2000. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 324:177–201, March.

Mehryar Mohri. 2005. Local grammar algorithms. In Antti Arppe, Lauri Carlson, Krister Lindèn, Jussi Piitulainen, Mickael Suominen, Martti Vainio, Hanna Westerlund, and Anssi Yli-Jyrä, editors, *Inquiries into Words, Constraints, and Contexts: Festschrift in Honour of Kimmo Koskenniemi on his 60th Birthday*, chapter 9, pages 84–93. CSLI Publications, Stanford University.

Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. 1999. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of UAI*, pages 467–475.

Anil Nelakanti, Cédric Archambeau, Julien Mairal, Francis Bach, Guillaume Bouchard, et al. 2013. Structured penalties for log-linear language models. In *Proceedings of EMNLP*, pages 233–243.

Neal Parikh and Stephen Boyd. 2013. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231.

Michael Paul and Jason Eisner. 2012. Implicitly intersecting weighted automata using dual decomposition. In *Proceedings of NAACL-HLT*, pages 232–242, Montreal, June.

Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California.

Fernando C. N. Pereira and Michael Riley. 1997. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*. MIT Press, Cambridge, MA.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of COLING-ACL*, pages 433–440, July.

Yuan Qi. 2005. *Extending Expectation Propagation for Graphical Models*. Ph.D. thesis, Massachusetts Institute of Technology, February.

Dana Ron, Yoram Singer, and Naftali Tishby. 1996. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149.

Mark W Schmidt and Kevin P Murphy. 2010. Convex structure learning in log-linear models: Beyond pairwise potentials. In *Proceedings of AISTATS*, pages 709–716.

Vesa Siivola, Teemu Hirsimäki, and Sami Virpioja. 2007. On growing and pruning Kneser-Ney smoothed n-gram models. *IEEE Transactions on Audio, Speech, and Language Processsing*, 15(5):1617–1624.

Andreas Stolcke. 1998. Entropy-based pruning of backoff language models. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274.

Ming Yuan and Yi Lin. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.