Dependency Parsing by Belief Propagation*

David A. Smith and **Jason Eisner**
Dept. of Computer Science, Johns Hopkins University
Baltimore, MD 21218, USA
{dasmith,eisner}@jhu.edu

Abstract

We formulate dependency parsing as a graphical model with the novel ingredient of global constraints. We show how to apply loopy belief propagation (BP), a simple and effective tool for *approximate* learning and inference. As a parsing algorithm, BP is both asymptotically and empirically efficient. Even with second-order features or latent variables, which would make exact parsing considerably slower or NP-hard, BP needs only $O(n^3)$ time with a small constant factor. Furthermore, such features significantly improve parse accuracy over exact first-order methods. Incorporating additional features would increase the runtime additively rather than multiplicatively.

1 Introduction

Computational linguists worry constantly about runtime. Sometimes we oversimplify our models, trading linguistic nuance for fast dynamic programming. Alternatively, we write down a better but intractable model and then use approximations. The CL community has often approximated using heavy pruning or reranking, but is beginning to adopt other methods from the machine learning community, such as Gibbs sampling, rejection sampling, and certain variational approximations.

We propose borrowing a different approximation technique from machine learning, namely, loopy belief propagation (BP). In this paper, we show that BP can be used to *train* and *decode* complex parsing models. Our approach calls a simpler parser as a subroutine, so it still exploits the useful, well-studied combinatorial structure of the parsing problem.[1]

2 Overview and Related Work

We wish to make a dependency parse's score depend on *higher-order* features, which consider arbitrary interactions among two or more edges in the parse (and perhaps also other latent variables such as part-of-speech tags or edge labels). Such features can help accuracy—as we show. Alas, they raise the polynomial runtime of projective parsing, and render non-projective parsing NP-hard (McDonald and Satta, 2007). Hence we seek approximations.

We will show how BP's "message-passing" discipline offers a principled way for *higher-order* features to incrementally adjust the numerical edge weights that are fed to a fast *first-order* parser. Thus the first-order parser is influenced by higher-order interactions among edges—but not asymptotically slowed down by considering the interactions itself.

BP's behavior in our setup can be understood intuitively as follows. Inasmuch as the first-order parser finds that edge e is probable, the higher-order features will kick in and discourage other edges e' to the extent that they prefer not to coexist with e.[2] Thus, the *next* call to the first-order parser assigns lower probabilities to parses that contain these e'. (The method is approximate because a first-order parser must equally penalize *all* parses containing e', even those that do not in fact contain e.)

This behavior is somewhat similar to parser stacking (Nivre and McDonald, 2008; Martins et al., 2008), in which a first-order parser derives some of its input features from the full 1-best output of another parser. In our method, a first-order parser derives such input features from its *own* previous full output (but probabilistic output rather than just 1-best). This circular process is *iterated* to convergence. Our method also permits the parse to interact cheaply with other variables. Thus first-order parsing, part-of-speech tagging, and other tasks on a common input could mutually influence one another.

Our method and its numerical details emerge naturally as an instance of the well-studied loopy BP algorithm, suggesting several future improvements

*This work was supported by the Human Language Technology Center of Excellence.

[1] As do constraint relaxation (Tromble and Eisner, 2006) and forest reranking (Huang, 2008). In contrast, generic NP-hard solution techniques like Integer Linear Programming (Riedel and Clarke, 2006) know nothing about optimal substructure.

[2] This may be reminiscent of adjusting a Lagrange multiplier on e' until some (hard) constraint is satisfied.

to accuracy (Yedidia et al., 2004; Braunstein et al., 2005) and efficiency (Sutton and McCallum, 2007).

Loopy BP has occasionally been used before in NLP, with good results, to handle non-local features (Sutton and McCallum, 2004) or joint decoding (Sutton et al., 2004). However, our application to parsing requires an innovation to BP that we explain in §5—a *global* constraint to enforce that the parse is a tree. The tractability of some such global constraints points the way toward applying BP to other computationally intensive NLP problems, such as syntax-based alignment of parallel text.

3 Graphical Models of Dependency Trees

3.1 Observed and hidden variables

To apply BP, we must formulate dependency parsing as a search for an optimal **assignment** to the variables of a graphical model. We encode a parse using the following variables:

Sentence. The n-word input sentence W is fully observed (not a lattice). Let $W = W_0 W_1 \cdots W_n$, where W_0 is always the special symbol ROOT.

Tags. If desired, the variables $T = T_1 T_2 \cdots T_n$ may specify tags on the n words, drawn from some tagset $\mathcal{T}$ (e.g., parts of speech). These variables are needed iff the tags are to be inferred jointly with the parse.

Links. The $O(n^2)$ boolean variables $\{L_{ij} : 0 \leq i \leq n, 1 \leq j \leq n, i \neq j\}$ correspond to the possible links in the dependency parse.[3] $L_{ij} = \mathsf{true}$ is interpreted as meaning that there exists a dependency link from parent $i \rightarrow$ child j.[4]

Link roles, etc. It would be straightforward to add other variables, such as a binary variable L_{irj} that is true iff there is a link $i \xrightarrow{r} j$ labeled with role r (e.g., AGENT, PATIENT, TEMPORAL ADJUNCT).

3.2 Markov random fields

We wish to define a probability distribution over all configurations, i.e., all joint assignments $\mathcal{A}$ to these variables. Our distribution is simply an undirected graphical model, or Markov random field (MRF):[5]

$$p(\mathcal{A}) \stackrel{\text{def}}{=} \frac{1}{Z} \prod_m F_m(\mathcal{A}) \quad (1)$$

specified by the collection of **factors** $F_m : \mathcal{A} \mapsto \mathbb{R}^{\geq 0}$. Each factor is a function that consults only a *subset* of $\mathcal{A}$. We say that the factor has **degree** d if it depends on the values of d variables in $\mathcal{A}$, and that it is **unary**, **binary**, **ternary**, or **global** if d is respectively 1, 2, 3, or unbounded (grows with n).

A factor function $F_m(\mathcal{A})$ may also depend freely on the observed variables—the input sentence W and a known (learned) parameter vector θ. For notational simplicity, we suppress these extra arguments when writing and drawing factor functions, and when computing their degree. In this treatment, these observed variables are not specified by $\mathcal{A}$, but instead are absorbed into the very definition of F_m.

§§3.3–3.4 lay out all of our parsing-specific factors; *their details are not too important*. In defining a factor F_m, we often state the circumstances under which it **fires**. These are the only circumstances that allow $F_m(\mathcal{A}) \neq 1$. When F_m does not fire, $F_m(\mathcal{A}) = 1$ and does not affect the product in (1).

3.3 List of hard constraints [skim on first reading]

A **hard** factor F_m fires only on parses $\mathcal{A}$ that *violate* some specified condition. It has value 0 on those parses, acting as a hard constraint to rule them out.

TREE. A hard global constraint on all the L_{ij} variables at once. It requires that exactly n of these variables be true, and that the corresponding links form a directed tree that is rooted at position 0.

PTREE. This stronger version of TREE requires further that the tree be **projective**. That is, it prohibits L_{ij} and $L_{k\ell}$ from both being true if $i \rightarrow j$ crosses $k \rightarrow \ell$. (These links are said to **cross** if one of k, ℓ is strictly between i and j while the other is strictly outside that range.)

EXACTLY1. A *family* of $O(n)$ hard global constraints, indexed by $1 \leq j \leq n$. EXACTLY1$_j$ requires that j have exactly one parent, i.e., exactly one of the L_{ij} variables must be true. Note that EXACTLY1 is implied by TREE or PTREE.

[3] "Links" are conventionally called edges, but we reserve the term "edge" for describing the graphical model's factor graph.

[4] We could have chosen a different representation with $O(n)$ integer variables $\{P_j : 1 \leq j \leq n\}$, writing $P_j = i$ instead of $L_{ij} = \mathsf{true}$. This representation can achieve the same asymptotic runtime for BP by using sparse messages, but some constraints and algorithms would be harder to explain.

[5] Our overall model is properly called a *dynamic* MRF, since we must construct different-size MRFs for input sentences of different lengths. Parameters are shared both across and within these MRFs, so that only finitely many parameters are needed.

ATMOST1. A weaker version. ATMOST1$_j$ requires j to have one or zero parents.

NAND. A *family* of hard binary constraints. NAND$_{ij,k\ell}$ requires that L_{ij} and $L_{k\ell}$ may not both be true. We will be interested in certain subfamilies.

NOT2. Shorthand for the family of $O(n^3)$ binary constraints $\{\text{NAND}_{ij,kj}\}$. These are collectively equivalent to ATMOST1, but expressed via a larger number of simpler constraints, which can make the BP approximation less effective (footnote 31).

NO2CYCLE. Shorthand for the family of $O(n^2)$ binary constraints $\{\text{NAND}_{ij,ji}\}$.

3.4 List of soft constraints [skim on first reading]

A **soft** factor F_m acts as a soft constraint that prefers some parses to others. In our experiments, it is always a log-linear function returning positive values:

$$F_m(\mathcal{A}) \stackrel{\text{def}}{=} \exp \sum_{h \in \text{features}(F_m)} \theta_h f_h(\mathcal{A}, W, m) \quad (2)$$

where θ is a learned, finite collection of weights and f is a corresponding collection of feature functions, some of which are used by F_m. (Note that f_h is permitted to consult the observed input W. It also sees which factor F_m it is scoring, to support reuse of a single feature function f_h and its weight θ_h by unboundedly many factors in a model.)

LINK. A family of unary soft factors that judge the links in a parse $\mathcal{A}$ individually. LINK$_{ij}$ fires iff $L_{ij} =$ true, and then its value depends on (i, j), W, and θ. Our experiments use the same features as McDonald et al. (2005).

A first-order (or "edge-factored") parsing model (McDonald et al., 2005) contains *only* LINK factors, along with a global TREE or PTREE factor. Though there are $O(n^2)$ link factors (one per L_{ij}), only n of them fire on any particular parse, since the global factor ensures that exactly n are true.

We'll consider various higher-order soft factors:

PAIR. A binary factor PAIR$_{ij,k\ell}$ fires with some value iff L_{ij} and $L_{k\ell}$ are both true. Thus, it penalizes or rewards a pair of links for being simultaneously present. This is a soft version of NAND.

GRAND. Shorthand for the family of $O(n^3)$ binary factors $\{\text{PAIR}_{ij,jk}\}$, which evaluate grandparent-parent-child configurations, $i \to j \to k$. For example, whether preposition j attaches to verb i might depend on its object k. Or in non-projective parsing, we might prefer (but not require) that a parent and child be on the same side of the grandparent.

SIB. Shorthand for the family of $O(n^3)$ binary factors $\{\text{PAIR}_{ij,ik}\}$, which judge whether two children of the same parent are compatible. E.g., a given verb may not like to have two noun children both to its left.[6] The children do not need to be adjacent.

CHILDSEQ. A family of $O(n)$ global factors. CHILDSEQ$_i$ scores i's *sequence* of children; hence it consults all variables of the form L_{ij}. If word 5 has children $2, 7, 9$ under $\mathcal{A}$, then CHILDSEQ$_i$ uses bigram models to score the left sequence #2# and right sequence #79# (where # is a boundary symbol).[7]

NOCROSS. A family of $O(n^2)$ global constraints. If the parent-to-j link crosses the parent-to-ℓ link, then NOCROSS$_{j\ell}$ fires with a value that depends only on j and ℓ. (If j and ℓ do not each have exactly one parent, NOCROSS$_{j\ell}$ fires with value 0; i.e., it incorporates EXACTLY1$_j$ and EXACTLY1$_\ell$.)[8]

TAG$_i$ is a unary factor that evaluates whether T_i's value is consistent with W (especially W_i).

TAGLINK$_{ij}$ is a ternary version of the LINK$_{ij}$ factor whose value depends on L_{ij}, T_i and T_j (i.e., its feature functions consult the tag variables to decide whether a link is likely). One could similarly enrich the other features above to depend on tags and/or link roles; TAGLINK is just an illustrative example.

TRIGRAM is a global factor that evaluates the tag sequence T according to a trigram model. It is a product of subfactors, each of which scores a trigram of adjacent tags T_{i-2}, T_{i-1}, T_i, possibly also considering the word sequence W (as in CRFs).

4 A Sketch of Belief Propagation

MacKay (2003, chapters 16 and 26) provides an excellent introduction to belief propagation, a gen-

[6] A similar binary factor could directly discourage giving the verb two SUBJECTs, if the model has variables for link roles.

[7] This follows the parameterization of a weighted split head-automaton grammar (Eisner and Satta, 1999). The score in this case is a product of subfactors of the form PAIR$_{5\#,52}$, PAIR$_{52,5\#}$ (left) and PAIR$_{5\#,57}$, PAIR$_{57,59}$, PAIR$_{59,5\#}$ (right).

[8] In effect, we have combined the $O(n^4)$ binary factors PAIR$_{ij,k\ell}$ into $O(n^2)$ groups, and made them more precise by multiplying in EXACTLYONE constraints (see footnote 31). This will permit $O(n^3)$ total computation if we are willing to sacrifice the ability of the PAIR weights to depend on i and k.

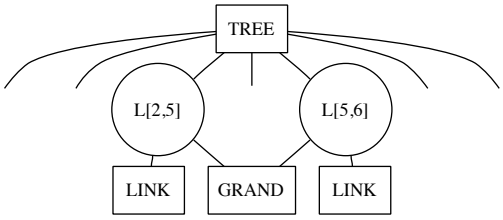

Figure 1: A fragment of a factor graph, illustrating a few of the unary, binary, and global factors that affect variables L_{25} and L_{56}. The GRAND factor induces a loop.

eralization of the forward-backward algorithm that is deeply studied in the graphical models literature (Yedidia et al., 2004, for example). We briefly sketch the method in terms of our parsing task.

4.1 Where BP comes from

The basic BP idea is simple. Variable L_{34} maintains a *distribution* over values true and false—a "belief"—that is periodically recalculated based on the current *distributions* at other variables.[9]

Readers familiar with Gibbs sampling can regard this as a kind of deterministic approximation. In Gibbs sampling, L_{34}'s *value* is periodically resampled based on the current *values* of other variables. Loopy BP works not with random samples but their expectations. Hence it is approximate but tends to converge much faster than Gibbs sampling will mix.

It is convenient to visualize an undirected **factor graph** (Fig. 1), in which each factor is connected to the variables it depends on. Many factors may connect to—and hence influence—a given variable such as L_{34}. If X is a variable *or* a factor, $\mathcal{N}(X)$ denotes its set of neighbors.

4.2 What BP accomplishes

Given an input sentence W and a parameter vector θ, the collection of factors F_m defines a probability distribution (1). The parser should determine the values of the individual variables. In other words, we would like to marginalize equation (1) to obtain the distribution $p(L_{34})$ over L_{34} = true vs. false, the distribution $p(T_4)$ over tags, etc.

If the factor graph is *acyclic*, then BP computes these marginal distributions exactly. Given an HMM, for example, BP reduces to the forward-backward algorithm.

BP's estimates of these distributions are called **beliefs about the variables**. BP also computes **beliefs about the factors**, which are useful in learning θ (see §7). E.g., if the model includes the factor TAGLINK$_{ij}$, which is connected to variables L_{ij}, T_i, T_j, then BP will estimate the marginal joint distribution $p(L_{ij}, T_i, T_j)$ over (boolean, tag, tag) triples.

When the factor graph has loops, BP's beliefs are usually not the true marginals of equation (1) (which are in general intractable to compute). Indeed, BP's beliefs may not be the true marginals of *any* distribution $p(\mathcal{A})$ over assignments, i.e., they may be globally inconsistent. All BP does is to incrementally adjust the beliefs till they are at least *locally* consistent: e.g., the beliefs at factors TAGLINK$_{ij}$ and TAGLINK$_{ik}$ must both imply[10] the same belief about variable T_i, their common neighbor.

4.3 The BP algorithm

This iterated negotiation among the factors is handled by **message passing** along the edges of the factor graph. A **message** to or from a variable is a (possibly unnormalized) probability distribution over the values of that variable.

The variable V sends a message to factor F, saying "My *other* neighboring factors G jointly suggest that I have posterior distribution $q_{V\to F}$ (assuming that they are sending me independent evidence)." Meanwhile, factor F sends messages to V, saying, "Based on my factor function and the messages received from my *other* neighboring variables U about *their* values (and assuming that those messages are independent), I suggest you have posterior distribution $r_{F\to V}$ over *your* values."

To be more precise, BP at each iteration k (until convergence) updates two kinds of messages:

$$q^{(k+1)}_{V\to F}(v) \;=\; \kappa \prod_{G\in\mathcal{N}(V),G\neq F} r^{(k)}_{G\to V}(v) \qquad (3)$$

from variables to factors, and

$$r^{(k+1)}_{F\to V}(v) = \kappa \sum_{\mathcal{A}\text{ s.t. }\mathcal{A}[V]=v} F(\mathcal{A}) \prod_{U\in\mathcal{N}(F),U\neq V} q^{(k)}_{U\to F}(\mathcal{A}[U]) \qquad (4)$$

[9] Or, more precisely—this is the tricky part—based on versions of those other distributions that do not factor in L_{34}'s reciprocal influence on *them*. This prevents (e.g.) L_{34} and T_3 from mutually reinforcing each other's existing beliefs.

[10] In the sense that marginalizing the belief $p(L_{ij}, T_i, T_j)$ at the factor yields the belief $p(T_i)$ at the variable.

from factors to variables. Each message is a probability distribution over values v of V, normalized by a scaling constant κ. Alternatively, messages may be left as unnormalized distributions, choosing $\kappa \neq 1$ only as needed to prevent over- or underflow. Messages are initialized to uniform distributions.

Whenever we wish, we may compute the beliefs at V and F:

$$b_{V\to}^{(k+1)}(v) \stackrel{\text{def}}{=} \kappa \prod_{G \in \mathcal{N}(V)} r_{G\to V}^{(k)}(v) \quad (5)$$

$$b_{F\to}^{(k+1)}(\mathcal{A}) \stackrel{\text{def}}{=} \kappa\, F(\mathcal{A}) \prod_{U \in \mathcal{N}(F)} q_{U\to F}^{(k)}(\mathcal{A}[U]) \quad (6)$$

These beliefs do not truly characterize the expected behavior of Gibbs sampling (§4.1), since the products in (5)–(6) make conditional independence assumptions that are valid only if the factor graph is acyclic. Furthermore, on cyclic ("loopy") graphs, BP might only converge to a local optimum (Weiss and Freedman, 2001), or it might not converge at all. Still, BP often leads to good, fast approximations.

5 Achieving Low Asymptotic Runtime

One iteration of standard BP simply updates all the messages as in equations (3)–(4): one message per edge of the factor graph.

Therefore, adding new factors to the model increases the runtime per iteration *additively*, by increasing the number of messages to update. We believe this is a compelling advantage over dynamic programming—in which new factors usually increase the runtime and space *multiplicatively* by exploding the number of distinct items.[11]

5.1 Propagators for local constraints

But how long does updating each message take? The runtime of summing over all assignments $\sum_{\mathcal{A}}$ in equation (4) may appear prohibitive. Crucially, however, $F(\mathcal{A})$ only depends on the values in $\mathcal{A}$ of F's its neighboring variables $\mathcal{N}(F)$. So this sum is proportional to a sum over *restricted* assignments to just those variables.[12]

For example, computing a message from $\text{TAGLINK}_{ij} \to T_i$ only requires iterating over all (boolean, tag, tag) triples.[13] The runtime to update that message is therefore $O(2 \cdot |\mathcal{T}| \cdot |\mathcal{T}|)$.

5.2 Propagators for global constraints

The above may be tolerable for a ternary factor. But how about global factors? EXACTLY1_j has n neighboring boolean variables: surely we cannot iterate over all 2^n assignments to these! TREE is even worse, with $2^{O(n^2)}$ assignments to consider. We will give specialized algorithms for handling these summations more efficiently.

A historical note is in order. Traditional constraint satisfaction corresponds to the special case of (1) where *all* factors F_m are hard constraints (with values in $\{0, 1\}$). In that case, loopy BP reduces to an algorithm for generalized arc consistency (Mackworth, 1977; Bessière and Régin, 1997; Dechter, 2003), and updating a factor's outgoing messages is known as **constraint propagation**. Régin (1994) famously introduced an efficient propagator for a global constraint, ALLDIFFERENT, by adapting combinatorial bipartite matching algorithms.

In the same spirit, we will demonstrate efficient propagators for our global constraints, e.g. by adapting combinatorial algorithms for *weighted parsing*. We are unaware of any previous work on global factors in sum-product BP, although for *max-product* BP,[14] Duchi et al. (2007) independently showed that a global 1-to-1 alignment constraint—a kind of weighted ALLDIFFERENT—permits an efficient propagator based on weighted bipartite matching.

5.3 Constraint propagators for parsing

Table 1 shows our asymptotic runtimes for all factors in §§3.3–3.4. Remember that if several of these

[11] For example, with unknown tags T, a model with PTREE+TAGLINK will take only $O(n^3 + n^2 g^2)$ time for BP, compared to $O(n^3 g^2)$ time for dynamic programming (Eisner & Satta 1999). Adding TRIGRAM, which is string-local rather than tree-local, will increase this only to $O(n^3 + n^2 g^2 + n g^3)$, compared to $O(n^3 g^6)$ for dynamic programming.

Even more dramatic, adding the SIB family of $O(n^3)$ $\text{PAIR}_{ij,ik}$ factors will add only $O(n^3)$ to the runtime of BP (Table 1). By contrast, the runtime of dynamic programming becomes exponential, because each item must record its headword's full *set* of current children.

[12] The constant of proportionality may be folded into κ; it is the number of assignments to the *other* variables.

[13] Separately for *each* value v of T_i, get v's probability by summing over assignments to (L_{ij}, T_i, T_j) s.t. $T_i = v$.

[14] Max-product replaces the sums in equations (3)–(4) with maximizations. This replaces the forward-backward algorithm with its Viterbi approximation.

factor family	degree (each)	runtime (each)	count	runtime (total)
TREE	$O(n^2)$	$O(n^3)$	1	$O(n^3)$
PTREE	$O(n^2)$	$O(n^3)$	1	$O(n^3)$
EXACTLY1	$O(n)$	$O(n)$	n	$O(n^2)$
ATMOST1	$O(n)$	$O(n)$	n	$O(n^2)$
NOT2	2	$O(1)$	$O(n^3)$	$O(n^3)$
NO2CYCLE	2	$O(1)$	$O(n^2)$	$O(n^2)$
LINK	1	$O(1)$	$O(n^2)$	$O(n^2)$
GRAND	2	$O(1)$	$O(n^3)$	$O(n^3)$
SIB	2	$O(1)$	$O(n^3)$	$O(n^3)$
CHILDSEQ	$O(n)$	$O(n^2)$	$O(n)$	$O(n^3)$
NOCROSS	$O(n)$	$O(n)$	$O(n^2)$	$O(n^3)$
TAG	1	$O(g)$	$O(n)$	$O(ng)$
TAGLINK	3	$O(g^2)$	$O(n^2)$	$O(n^2g^2)$
TRIGRAM	$O(n)$	$O(ng^3)$	1	$O(ng^3)$

Table 1: Asymptotic runtimes of the propagators for various factors (where n is the sentence length and g is the size of the tag set $\mathcal{T}$). An iteration of standard BP propagates through each factor once. Running a factor's propagator will update all of its outgoing messages, based on its current incoming messages.

factors are included, the total runtime is *additive*.[15]

Propagating the local factors is straightforward (§5.1). We now explain how to handle the global factors. *Our main trick is to work backwards from marginal beliefs.* Let F be a factor and V be one of its neighboring variables. At any time, F has a marginal belief about V (see footnote 10),

$$b_{F\rightarrow}^{(k+1)}(V=v) = \sum_{\mathcal{A} \text{ s.t. } \mathcal{A}[V]=v} b_{F\rightarrow}^{(k+1)}(\mathcal{A}) \quad (7)$$

a sum over (6)'s products of incoming messages. By the definition of $r_{F\rightarrow V}$ in (4), and distributivity, we can also express the marginal belief (7) as a pointwise product of outgoing and incoming messages[16]

$$b_{F\rightarrow}^{(k+1)}(V=v) = r_{F\rightarrow V}^{(k+1)}(v) \cdot q_{V\rightarrow F}^{(k)}(v) \quad (8)$$

up to a constant. If we can quickly sum up the marginal belief (7), then (8) says we can divide out *each* particular incoming message $q_{V\rightarrow F}^{(k)}$ in turn to obtain its corresponding outgoing message $r_{F\rightarrow V}^{(k+1)}$.

[15]We may ignore the cost of propagators at the variables. Each outgoing message from a variable can be computed in time proportional to its size, which may be amortized against the cost of generating the corresponding incoming message.

[16]E.g., the familiar product of forward and backward messages that is used to extract posterior marginals from an HMM.

Note that the marginal belief and both messages are unnormalized distributions over values v of V. F and k are clear from context below, so we simplify the notation so that (7)–(8) become

$$b(V=v) \;=\; \sum_{\mathcal{A} \text{ s.t. } \mathcal{A}[V]=v} b(\mathcal{A}) \;=\; r_V(v) \cdot q_V(v)$$

TRIGRAM must sum over assignments to the tag sequence T. The belief (6) in a given assignment is a product of trigram scores (which play the role of transition weights) and incoming messages q_{T_j} (playing the role of emission weights). The marginal belief (7) needed above, $b(T_i = t)$, is found by summing over assignments where $T_i = t$. All marginal beliefs are computed together in $O(ng^3)$ total time by the forward-backward algorithm.[17]

EXACTLY1$_j$ is a *sparse* hard constraint. Even though there are 2^n assignments to its n neighboring variables $\{L_{ij}\}$, the factor function returns 1 on only n assignments and 0 on the rest. In fact, for a given i, $b(L_{ij} = \mathsf{true})$ in (7) is defined by (6) to have exactly one non-zero summand, in which $\mathcal{A}$ puts $L_{ij} = \mathsf{true}$ and all other $L_{i'j} = \mathsf{false}$. We compute the marginal beliefs (7) for all i together in $O(n)$ total time:

1. Pre-compute $\pi \stackrel{\text{def}}{=} \prod_i q_{L_{ij}}(\mathsf{false})$.[18]
2. For each i, compute the marginal belief $b(L_{ij} = \mathsf{true})$ as $\pi \cdot \bar{q}_{L_{ij}}$, where $\bar{q}_{L_{ij}} \in \mathbb{R}$ denotes the odds ratio $q_{L_{ij}}(\mathsf{true})/q_{L_{ij}}(\mathsf{false})$.[19]
3. The partition function $b()$ denotes $\sum_{\mathcal{A}} b(\mathcal{A})$; compute it in this case as $\sum_i b(L_{ij} = \mathsf{true})$.
4. For each i, compute $b(L_{ij} = \mathsf{false})$ by subtraction, as $b() - b(L_{ij} = \mathsf{true})$.

TREE and PTREE must sum over assignments to the $O(n^2)$ neighboring variables $\{L_{ij}\}$. There are now exponentially many non-zero summands in (7), those in which $\mathcal{A}$ corresponds to a valid tree.

[17]Which is itself an exact BP algorithm, but on a different graph—a junction tree formed from the graph of TRIGRAM subfactors. Each variable in the junction tree is a *bigram*. If we had simply replaced the global TRIGRAM factor with its subfactors in the full factor graph, we would have had to resort to Generalized BP (Yedidia et al., 2004) to obtain the same exact results.

[18]But taking $\pi = 1$ gives the same results, up to a constant.

[19]As a matter of implementation, this odds ratio $\bar{q}_{L_{ij}}$ can be used to represent the incoming message $q_{L_{ij}}$ everywhere.

Nonetheless, we can follow the same approach as for EXACTLY1. Steps 1 and 4 are modified to iterate over all i, j such that L_{ij} is a variable. In step 3, the partition function $\sum_{\mathcal{A}} b(\mathcal{A})$ is now π times the total weight of all trees, where the weight of a given tree is the product of the $\bar{q}_{L_{ij}}$ values of its n edges. In step 2, the marginal belief $b(L_{ij} = \mathsf{true})$ is now π times the total weight of all trees having edge $i \to j$.

We perform these combinatorial sums *by calling a first-order parsing algorithm,* with edge weights $\bar{q}_{ij}$. Thus, as outlined in §2, a first-order parser is called each time we propagate through the global TREE or PTREE constraint, using edge weights that include the first-order LINK factors but also multiply in any current messages from higher-order factors.

The parsing algorithm simultaneously computes the partition function $b()$ and all $O(n^2)$ marginal beliefs $b(L_{ij} = \mathsf{true})$. For PTREE (projective), it is the inside-outside version of a dynamic programming algorithm (Eisner, 1996). For TREE (non-projective), Koo et al. (2007) and Smith and Smith (2007) show how to employ the matrix-tree theorem. In both cases, the total time is $O(n^3)$.[20]

NOCROSS$_{j\ell}$ must sum over assignments to $O(n)$ neighboring variables $\{L_{ij}\}$ and $\{L_{k\ell}\}$. The non-zero summands are assignments where j and ℓ each have exactly one parent. At step 1, $\pi \stackrel{\text{def}}{=} \prod_i q_{L_{ij}}(\mathsf{false}) \cdot \prod_k q_{L_{k\ell}}(\mathsf{false})$. At step 2, the marginal belief $b(L_{ij} = \mathsf{true})$ sums over the n non-zero assignments containing $i \to j$. It is $\pi \cdot \bar{q}_{L_{ij}} \cdot \sum_k \bar{q}_{L_{k\ell}} \cdot \text{PAIR}_{ij,k\ell}$, where $\text{PAIR}_{ij,k\ell}$ is $x_{j\ell}$ if $i \to j$ crosses $k \to \ell$ and is 1 otherwise. $x_{j\ell}$ is some factor value defined by equation (2) to penalize or reward the crossing. Steps 3–4 are just as in EXACTLY1$_j$.

The question is how to compute $b(L_{ij} = \mathsf{true})$ for each i in only $O(1)$ time,[21] so that we can propagate each of the $O(n^2)$ NOCROSS$_{j\ell}$ in $O(n)$ time. This is why we allowed $x_{j\ell}$ to depend only on j, ℓ. We can rewrite the sum $b(L_{ij} = \mathsf{true})$ as

$$\pi \cdot \bar{q}_{L_{ij}} \cdot (x_{j\ell} \cdot \sum_{\text{crossing } k} \bar{q}_{L_{k\ell}} \; + \; 1 \cdot \sum_{\text{noncrossing } k} \bar{q}_{L_{k\ell}}) \quad (9)$$

To find this in $O(1)$ time, we precompute for each ℓ an array of partial sums $Q_\ell[s,t] \stackrel{\text{def}}{=} \sum_{s \le k \le t} \bar{q}_{L_{k\ell}}$. Since $Q_\ell[s,t] = Q_\ell[s,t-1] + \bar{q}_{L_{t\ell}}$, we can compute each entry in $O(1)$ time. The total precomputation time over all ℓ, s, t is then $O(n^3)$, with the array Q_ℓ shared across all factors NOCROSS$_{j'\ell}$. The crossing sum is respectively $Q_\ell[0, i-1] + Q_\ell[j+1, n]$, $Q_\ell[i+1, j-1]$, or 0 according to whether $\ell \in (i, j)$, $\ell \notin [i, j]$, or $\ell = i$.[22] The non-crossing sum is $Q_\ell[0, n]$ minus the crossing sum.

CHILDSEQ$_i$, like TRIGRAM, is propagated by a forward-backward algorithm. In this case, the algorithm is easiest to describe by replacing CHILDSEQ$_i$ in the factor graph by a collection of local subfactors, which pass messages in the ordinary way.[23] Roughly speaking,[24] at each $j \in [1, n]$, we introduce a new variable C_{ij}—a hidden state whose value is the position of i's previous child, if any (so $0 \le C_{ij} < j$). So the ternary subfactor on $(C_{ij}, L_{ij}, C_{i,j+1})$ has value 1 if $L_{ij} = \mathsf{false}$ and $C_{i,j+1} = C_{i,j}$; a sibling-bigram score ($\text{PAIR}_{iC_{ij},iC_{i,j+1}}$) if $L_{ij} = \mathsf{true}$ and $C_{i,j+1} = j$; and 0 otherwise. The sparsity of this factor, which is 0 almost everywhere, is what gives CHILDSEQ$_i$ a total runtime of $O(n^2)$ rather than $O(n^3)$. It is equivalent to forward-backward on an HMM with n observations (the L_{ij}) and n states per observation (the C_j), with a *deterministic* (thus sparse) transition function.

6 Decoding Trees

BP computes local beliefs, e.g. the conditional probability that a link L_{ij} is present. But if we wish to output a single well-formed dependency tree, we need to find a single assignment to all the $\{L_{ij}\}$ that satisfies the TREE (or PTREE) constraint.

Our final belief about the TREE *factor* is a distribution over such assignments, in which a tree's probability is proportional to the probability of its edge weights $\bar{q}_{L_{ij}}$ (incoming messages). We could simply return the mode of this distribution (found by using a 1-best first-order parser) or the k-best trees, or take samples.

[20] A dynamic algorithm could incrementally update the outgoing messages if only a few incoming messages have changed (as in asynchronous BP). In the case of TREE, dynamic matrix inverse allows us to update any row or column (i.e., messages from all parents or children of a given word) and find the new inverse in $O(n^2)$ time (Sherman and Morrison, 1950).

[21] Symmetrically, we compute $b(L_{k\ell} = \mathsf{true})$ for each k.

[22] There are no NOCROSS$_{j\ell}$ factors with $\ell = j$.

[23] Still we treat CHILDSEQ$_i$ as a global factor and compute all its correct outgoing messages on a *single* BP iteration, via *serial* forward and backward sweeps through the subfactors. Handling the subfactors in parallel, (3)–(4), would need $O(n)$ iterations.

[24] Ignoring the treatment of boundary symbols "#" (see §3.4).

In our experiments, we actually take the edge weights to be not the messages $\bar{q}_{L_{ij}}$ from the links, but the full beliefs $\bar{b}_{L_{ij}}$ at the links (where $\bar{b}_{L_{ij}} \stackrel{\text{def}}{=} \log b_{L_{ij}}(\mathsf{true})/b_{L_{ij}}(\mathsf{false})$). These are passed into a fast algorithm for maximum spanning tree (Tarjan, 1977) or maximum projective spanning tree (Eisner, 1996). This procedure is equivalent to minimum Bayes risk (MBR) parsing (Goodman, 1996) with a dependency accuracy loss function.

Notice that the above decoding approaches do not enforce any hard constraints other than TREE in the final output. In addition, they only recover values of the L_{ij} variables. They marginalize over other variables such as tags and link roles. This solves the problem of "nuisance" variables (which merely fragment probability mass among refinements of a parse). On the other hand, it may be undesirable for variables whose values we desire to recover.[25]

7 Training

Our training method also uses beliefs computed by BP, but at the *factors*. We choose the weight vector θ to maximize the log-probability under (1) of training data (regularizing only by early stopping, §8.3). If all variables are observed in training, this objective function is *convex* (as for any log-linear model).

The difficult step in computing the gradient of our objective is finding $\nabla_\theta \log Z$, where Z in equation (1) is the normalizing constant (partition function) that sums over all assignments $\mathcal{A}$. (Recall that Z, like each F_m, depends implicitly on W and θ.) As usual for log-linear models,

$$\nabla_\theta \log Z = \sum_m \mathbf{E}_{p(\mathcal{A})}[\nabla_\theta F_m(\mathcal{A})] \qquad (10)$$

Since $\nabla_\theta F_m(\mathcal{A})$ only depends on the assignment $\mathcal{A}$'s values for variables that are connected to F_m in the factor graph, its expectation under $p(\mathcal{A})$ depends only on the marginalization of $p(\mathcal{A})$ to those variables jointly. Fortunately, BP provides an estimate of that marginal distribution, namely, its belief (6) about the factor F_m, given W and θ (§4.2).[26]

Note that the hard constraints do not depend on θ at all; so their summands in equation (10) will be 0.

We employ stochastic gradient descent (Bottou, 2003), since this does not require us to compute the objective function itself but only to (approximately) estimate its gradient as explained above. Alternatively, given any of the MAP decoding procedures from §6, we could use an error-driven learning method such as the perceptron or MIRA.[27]

8 Experiments

We asked: (1) For projective parsing, where higher-order factors have traditionally been incorporated into slow but exact dynamic programming (DP), what are the comparative speed and quality of the BP approximation? (2) How helpful are such higher-order factors—particularly for non-projective parsing, where BP is needed to make them tractable? (3) Do our global constraints (e.g., TREE) contribute to the goodness of BP's approximation?

[25] An alternative is to attempt to find the most probable ("MAP") assignment to all variables—using the max-product algorithm (footnote 14) or one of its recent variants. The estimated marginal beliefs become "max marginals," which assess the 1-best assignment consistent with each value of the variable.

We can indeed build max-product propagators for our global constraints. PTREE still propagates in $O(n^3)$ time: simply change the first-order parser's semiring (Goodman, 1999) to use max instead of sum. TREE requires $O(n^4)$ time: it seems that the $O(n^2)$ max marginals must be computed separately, each requiring a separate call to an $O(n^2)$ maximum spanning tree algorithm (Tarjan, 1977).

If max-product BP converges, we may simply output each variable's favorite value (according to its belief), if unique. However, max-product BP tends to be unstable on loopy graphs, and we may not wish to wait for full convergence in any case. A more robust technique for extracting an assignment is to mimic Viterbi decoding, and "follow backpointers" of the max-product computation along some spanning subtree of the factor graph.

A slower but potentially more stable alternative is deterministic annealing. Replace each factor $F_m(\mathcal{A})$ with $F_m(\mathcal{A})^{1/T}$, where $T > 0$ is a **temperature**. As $T \to 0$ ("quenches"), the distribution (1) retains the same mode (the MAP assignment), but becomes more sharply peaked at the mode, and sum-product BP approaches max-product BP. Deterministic annealing runs sum-product BP while gradually reducing T toward 0 as it iterates. By starting at a high T and reducing T slowly, it often manages in practice to find a good local optimum. We may then extract an assignment just as we do for max-product.

[26] This approximates the estimate that would be built up more slowly by Gibbs sampling. One could use coarser estimates at earlier stages of training, by running fewer iterations of BP.

[27] The BP framework makes it tempting to extend an MRF model with various sorts of latent variables, whose values are not specified in training data. It is straightforward to train under these conditions. When counting which features fire on a training parse or (for error-driven training) on an current erroneous parse, we can find *expected* counts if these parses are not fully observed, by using BP to sum over latent variables.

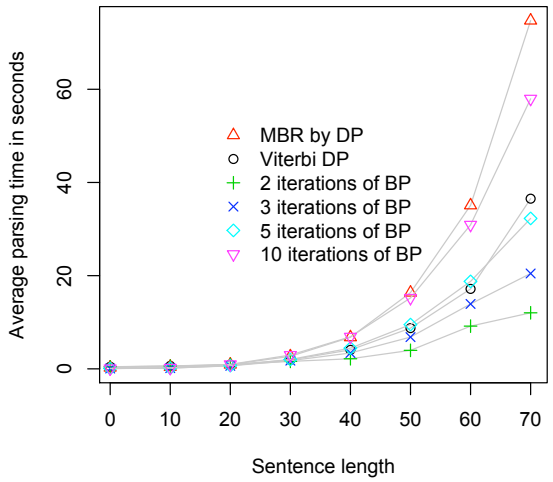

Figure 2: Runtime of BP parser on various sentence lengths compared to $O(n^4)$ dynamic programming.

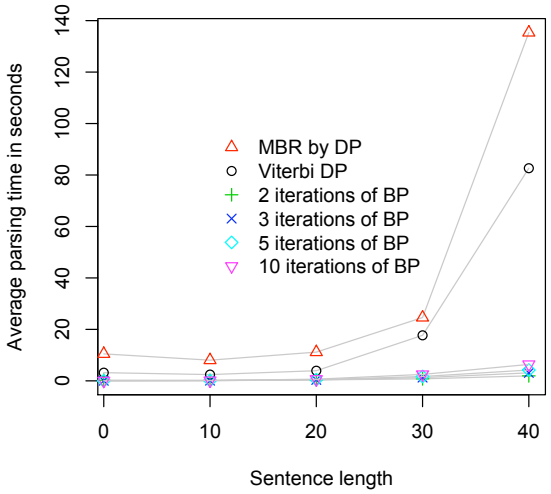

Figure 3: Runtime of BP parser on various sentence lengths compared to $O(n^5)$ dynamic programming. DP is so slow for length > 45 that we do not even show it.

8.1 Data

We trained and tested on three languages from the CoNLL Dependency Parsing Shared Task (Nivre et al., 2007). The English data for that task were converted from the Penn Treebank to dependencies using a trace-recovery algorithm that induced some very slight non-projectivity—about 1% of links crossed other links. Danish is a slightly more non-projective language (3% crossing links). Dutch is the most non-projective language in the corpus (11%). In all cases, the test input W consists of part-of-speech-tagged words (bearing both "coarse" and "fine" tags), so T variables were not used.

8.2 Features

Although BP makes it cheap to incorporate many non-local features and latent variables at once, we kept our models relatively simple in this paper.

Our first-order LINK$_{ij}$ factors replicate McDonald et al. (2005). Following equation (2), they are defined using binary features that look at words i and j, the distance $j - i$, and the tags (provided in W) of words at, around, and between i and j.

Our second-order features are similar. In the GRAND factors, features fire for particular triples of tags and of coarse tags. A feature also fires if the grandparent falls between the child and parent, inducing crossing dependency links. The CHILDSEQ factors included features for tags, and likewise coarse tags, on adjacent sibling pairs and parent-sibling-sibling triples. Each of these features also have versions that were conjoined with link direction—pairs of directions in the grandparent case—or with signed link length of the child or farther sibling. Lengths were binned per McDonald et al. (2005). The NOCROSS$_{j\ell}$ factors consider the tag and coarse tag attributes of the two child words j and ℓ, separately or jointly.

8.3 Experimental procedures

We trained all models using stochastic gradient descent (§7). SGD initialized $\vec{\theta} = 0$ and ran for 10 consecutive passes over the data; we picked the stopping point that performed best on held-out data.

When comparing runtimes for projective parsers, we took care to produce comparable implementations. All beliefs and dynamic programming items were stored and indexed using the high-level Dyna language,[28] while all inference and propagation was written in C++. The BP parser averaged 1.8 seconds per sentence for non-projective parsing and 1.5 seconds per sentence for projective parsing (1.2 and 0.9 seconds/sentence for ≤ 40 words), using our standard setup, which included five iterations of BP and the final MBR tree decoding pass.

In our tables, we boldface the best result in each column along with any results that are not significantly worse (paired permutation test, $p < .05$).

[28] This dominates runtime, and probably slows down all our parsers by a factor of 4–11 owing to known inefficiencies in the Dyna prototype we used (Eisner et al., 2005).

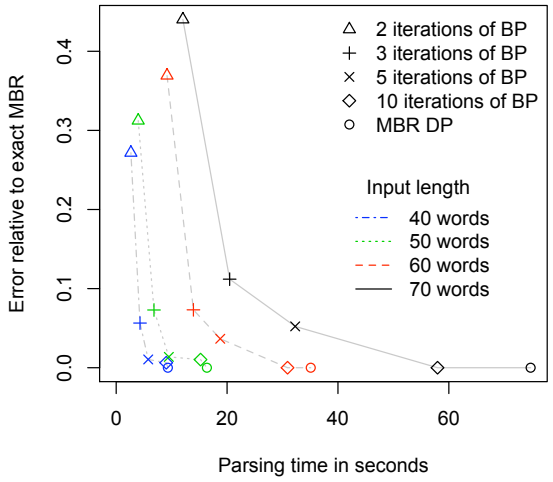

Figure 4: Runtime vs. search error after different numbers of BP iterations. This shows the simpler model of Fig. 2, where DP is still relatively fast.

8.4 Faster higher-order projective parsing

We built a first-order projective parser—one that uses only factors PTREE and LINK—and then compared the cost of incorporating second-order factors, GRAND and CHILDSEQ, by BP versus DP.[29]

Under DP, the first-order runtime of $O(n^3)$ is increased to $O(n^4)$ with GRAND, and to $O(n^5)$ when we add CHILDSEQ as well. BP keeps runtime down to $O(n^3)$—although with a higher constant factor, since it takes several rounds to converge, and since it computes more than just the best parse.[30]

Figures 2–3 compare the empirical runtimes for various input sentence lengths. With only the GRAND factor, exact DP can still find the Viterbi parse (though not the MBR parse[30]) faster than ten iterations of the asymptotically better BP (Fig. 2), at least for sentences with $n \leq 75$. However, once we add the CHILDSEQ factor, BP is always faster—dramatically so for longer sentences (Fig. 3). More complex models would widen BP's advantage.

Fig. 4 shows the tradeoff between runtime and search error of BP in the former case (GRAND only). To determine BP's search error at finding the MBR parse, we measured its dependency accuracy not against the gold standard, but against the optimal MBR parse under the model, which DP is able to find. After 10 iterations, the overall macro-averaged search error compared to $O(n^4)$ DP MBR is 0.4%; compared to $O(n^5)$ (not shown), 2.4%. More BP iterations may help accuracy. In future work, we plan to compare BP's speed-accuracy curve on more complex projective models with the speed-accuracy curve of *pruned* or *reranked* DP.

[29] We trained these parsers using exact DP, using the inside-outside algorithm to compute equation (10). The training and test data were English, and for this section we filtered out sentences with non-projective links.

[30] Viterbi parsing in the log domain only needs the $(\max, +)$ semiring, whereas both BP and any MBR parsing must use the slower $(+, \log+)$ so that they can compute marginals.

		Danish	Dutch	English
(a)	TREE+LINK	85.5	87.3	88.6
	+NOCROSS	86.1	88.3	89.1
	+GRAND	86.1	**88.6**	89.4
	+CHILDSEQ	**86.5**	**88.5**	90.1
(b)	Proj. DP	86.0	84.5	**90.2**
	+hill-climbing	86.1	87.6	**90.2**

Table 2: (a) Percent unlabeled dependency accuracy for various non-projective BP parsers (5 iterations only), showing the cumulative contribution of different features. (b) Accuracy for an projective DP parser with all features. For relatively non-projective languages (Danish and especially Dutch), the exact projective parses can be improved by non-projective hill-climbing—but in those cases, just running our non-projective BP is better and faster.

8.5 Higher-order non-projective parsing

The BP approximation can be used to improve the accuracy of *non*-projective parsing by adding higher-order features. These would be NP-hard to incorporate exactly; DP cannot be used.

We used BP with a non-projective TREE factor to train conditional log-linear parsing models of two highly non-projective languages, Danish and Dutch, as well as slightly non-projective English (§8.1). In all three languages, the first-order non-projective parser greatly overpredicts the number of crossing links. We thus added NOCROSS factors, as well as GRAND and CHILDSEQ as before. All of these significantly improve the first-order baseline, though not necessarily cumulatively (Table 2).

Finally, Table 2 compares loopy BP to a previously proposed "hill-climbing" method for approximate inference in non-projective parsing McDonald and Pereira (2006). Hill-climbing decodes our richest non-projective model by finding the best *projective* parse under that model—using slow, *higher-order* DP—and then greedily modifies words' parents until the parse score (1) stops improving.

BP for non-projective languages is much faster and more accurate than the hill-climbing method.

Decoding	Danish	Dutch	English
NOT2	81.8 (76.7)	83.3 (75.0)	87.5 (66.4)
ATMOST1	85.4 (82.2)	**87.3** (86.3)	88.5 (84.6)
EXACTLY1	**85.7** (85.0)	87.0 (86.7)	88.6 (86.0)
+ NO2CYCLE	85.0 (**85.2**)	86.2 (86.7)	88.5 (86.2)
TREE	**85.5** (**85.5**)	**87.3** (**87.3**)	88.6 (**88.6**)
PTREE	**85.8**	83.9	**88.8**

Table 3: After training a non-projective first-order model with TREE, decoding it with weaker constraints is asymptotically faster (except for NOT2) but usually harmful. (Parenthetical numbers show that the harm is compounded if the weaker constraints are used in training as well; even though this matches training to test conditions, it may suffer more from BP's approximate gradients.) Decoding the TREE model with the even stronger PTREE constraint can actually be helpful for a more projective language. All results use 5 iterations of BP.

Also, hill-climbing only produces an (approximate) 1-best parse, but BP also obtains (approximate) marginals of the distribution over *all* parses.

8.6 Importance of global hard constraints

Given the BP architecture, do we even need the hard TREE constraint? Or would it suffice for more local hard constraints to negotiate locally via BP?

We investigated this for non-projective first-order parsing. Table 3 shows that global constraints are indeed important, and that it is essential to use TREE during training. At test time, the weaker but still global EXACTLY1 may suffice (followed by MBR decoding to eliminate cycles), for total time $O(n^2)$.

Table 3 includes NOT2, which takes $O(n^3)$ time, merely to demonstrate how the BP approximation becomes more accurate for training and decoding when we join the simple NOT2 constraints into more global ATMOST1 constraints. This does not change the distribution (1), but makes BP enforce stronger local consistency requirements at the factors, relying less on independence assumptions. In general, one can get better BP approximations by replacing a group of factors $F_m(\mathcal{A})$ with their product.[31]

The above experiments concern *gold-standard* accuracy under a given *first-order*, *non-projective* model. Flipping all three of these parameters for Danish, we confirmed the pattern by instead measuring *search error* under a *higher-order*, *projective* model (PTREE+LINK+GRAND), when PTREE was weakened during decoding. Compared to the MBR parse under that model, the search errors from decoding with weaker hard constraints were 2.2% for NOT2, 2.1% for EXACTLY1, 1.7% for EXACTLY1 + NO2CYCLE, and 0.0% for PTREE.

9 Conclusions and Future Work

We showed how to coordinate a fast 1st-order parser with higher-order features, by "hiding" it within one step of a general approximate inference algorithm for graphical models—loopy belief propagation.

This improves *non-projective* dependency accuracy via features that would make exact parsing intractable. For *projective* parsing, it greatly speeds up dynamic programming—in theory and in practice—at the cost of small amounts of search error.

We are interested in extending these ideas to phrase-structure and lattice parsing, and in trying other higher-order features, such as those used in parse reranking (Charniak and Johnson, 2005; Huang, 2008) and history-based parsing (Nivre and McDonald, 2008). We could also introduce new variables, e.g., nonterminal refinements (Matsuzaki et al., 2005), or secondary links M_{ij} (not constrained by TREE/PTREE) that augment the parse with representations of control, binding, etc. (Sleator and Temperley, 1993; Buch-Kromann, 2006).

Other parsing-like problems that could be attacked with BP appear in syntax-based machine translation. Decoding is very expensive with a synchronous grammar composed with an n-gram language model (Chiang, 2007)—but our footnote 11 suggests that BP might incorporate a language model rapidly. String alignment with synchronous grammars is quite expensive even for simple synchronous formalisms like ITG (Wu, 1997)—but Duchi et al. (2007) show how to incorporate bipartite matching into max-product BP.

Finally, we can take advantage of improvements to BP proposed in the context of other applications. For example, instead of updating all messages in parallel at every iteration, it is empirically faster to serialize updates using a priority queue (Elidan et al., 2006; Sutton and McCallum, 2007).[32]

[31] In the limit, one could replace the product (1) with a single all-purpose factor; then BP would be exact—but slow. (In constraint satisfaction, joining constraints similarly makes arc consistency slower but better at eliminating impossible values.)

[32] These methods need alteration to handle our global propagators, which do update all their outgoing messages at once.

References

C. Bessière and J.-C. Régin. 1997. Arc consistency for general constraint networks: preliminary results. In *IJCAI*, pages 398–404.

L. Bottou. 2003. Stochastic learning. In *Advanced Lectures in Machine Learning*, pages 146–168. Springer.

A. Braunstein, M. Mezard, and R. Zecchina. 2005. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27:201–226.

M. Buch-Kromann. 2006. *Discontinuous Grammar. A Model of Human Parsing and Language Acquisition"*. Dr.ling.merc. dissertation, Copenhagen Business School.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL*, pages 173–180.

D. Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

R. Dechter. 2003. *Constraint Processing*. Morgan Kaufmann.

J. Duchi, D. Tarlow, G. Elidan, and D. Koller. 2007. Using combinatorial optimization within max-product belief propagation. In *NIPS 2006*, pages 369–376.

J. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *ACL*, pages 457–480.

J. Eisner, E. Goldlust, and N. A. Smith. 2005. Compiling comp ling: Weighted dynamic programming and the dyna language. In *HLT-EMNLP*, pages 281–290.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *COLING*.

G. Elidan, I. McGraw, and D. Koller. 2006. Residual belief propagation: Informed scheduling for asynchronous message passing. In *UAI*.

J. T. Goodman. 1996. Parsing algorithms and metrics. In *ACL*, pages 177–183.

J. Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

L. Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *ACL*, pages 586–594.

T. Koo, A. Globerson, X. Carreras, and M. Collins. 2007. Structured prediction models via the Matrix-Tree Theorem. In *EMNLP-CoNLL*.

D. MacKay. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge.

A. Mackworth. 1977. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118.

A. F. T. Martins, D. Das, N. A. Smith, and E. P. Xing. 2008. Stacking dependency parsers. In *EMNLP*.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2005. Probabilistic CFG with latent annotations. In *ACL*, pages 75–82.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*.

R. McDonald and G. Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *IWPT*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *ACL*.

J. Nivre and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *ACL*.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*.

J.-C. Régin. 1994. A filtering algorithm for constraints of difference in CSPs. In *AAAI*, pages 362–367.

S. Riedel and J. Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *EMNLP*, pages 129–137.

J. Sherman and W. J. Morrison. 1950. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Ann. Math. Stat.*, 21:124–127.

D. Sleator and D. Temperley. 1993. Parsing English with a link grammar. In *IWPT*, pages 277–291, August.

D. A. Smith and N. A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *EMNLP-CoNLL*.

C. Sutton and A. McCallum. 2004. Collective segmentation and labeling of distant entities in information extraction. In *ICML Workshop on Statistical Relational Learning*.

C. Sutton and A. McCallum. 2007. Improved dynamic schedules for belief propagation. In *UAI*.

C. Sutton, K. Rohanimanesh, and A. McCallum. 2004. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *ICML*.

R. E. Tarjan. 1977. Finding optimum branchings. *Networks*, 7:25–35.

R. W. Tromble and J. Eisner. 2006. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *HLT-NAACL*, pages 423–430.

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

D. Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *CL*, 23(3):377–404.

J. S. Yedidia, W. T. Freeman, and Y. Weiss. 2004. Constructing free-energy approximations and generalized belief approximation algorithms. MERL TR2004-040, Mitsubishi Electric Research Laboratories.