Approximate PCFG Parsing Using Tensor Decomposition

Shay B. Cohen
Department of Computer Science
Columbia University, USA
scohen@cs.columbia.edu

Giorgio Satta
Department of Information Engineering
University of Padua, Italy
satta@dei.unipd.it

Michael Collins
Department of Computer Science
Columbia University, USA
mcollins@cs.columbia.edu

Abstract

We provide an approximation algorithm for PCFG parsing, which asymptotically improves time complexity with respect to the input grammar size, and prove upper bounds on the approximation quality. We test our algorithm on two treebanks, and get significant improvements in parsing speed.

1 Introduction

The problem of speeding-up parsing algorithms based on probabilistic context-free grammars (PCFGs) has received considerable attention in recent years. Several strategies have been proposed, including beam-search, best-first and A*. In this paper we focus on the standard approach of approximating the source PCFG in such a way that parsing accuracy is traded for efficiency.

Nederhof (2000) gives a thorough presentation of old and novel ideas for approximating non-probabilistic CFGs by means of finite automata, on the basis of specialized preprocessing of self-embedding structures. In the probabilistic domain, approximation by means of regular grammars is also exploited by Eisner and Smith (2005), who filter long-distance dependencies on-the-fly.

Beyond finite automata approximation, Charniak et al. (2006) propose a coarse-to-fine approach in which an approximated (not necessarily regular) PCFG is used to construct a parse forest for the input sentence. Some statistical parameters are then computed on such a structure, and exploited to filter parsing with the non-approximated grammar. The approach can also be iterated at several levels. In the non-probabilistic setting, a similar filtering approach was also proposed by Boullier (2003), called "guided parsing."

In this paper we rely on an algebraic formulation of the inside-outside algorithm for PCFGs, based on a tensor formulation developed for latent-variable PCFGs in Cohen et al. (2012). We combine the method with known techniques for tensor decomposition to approximate the source PCFG, and develop a novel algorithm for approximate PCFG parsing. We obtain improved time upper bounds with respect to the input grammar size for PCFG parsing, and provide error upper bounds on the PCFG approximation, in contrast with existing heuristic methods.

2 Preliminaries

This section introduces the special representation for probabilistic context-free grammars that we adopt in this paper, along with the decoding algorithm that we investigate. For an integer $i \geq 1$, we let $[i] = \{1, 2, \ldots, i\}$.

2.1 Probabilistic Context-Free Grammars

We consider context-free grammars (CFGs) in Chomsky normal form, and denote them as $(\mathcal{N}, \mathcal{L}, \mathcal{R})$ where:

- $\mathcal{N}$ is the finite set of nonterminal symbols, with $m = |\mathcal{N}|$, and $\mathcal{L}$ is the finite set of words (lexical tokens), with $\mathcal{L} \cap \mathcal{N} = \emptyset$ and with $n = |\mathcal{L}|$.
- $\mathcal{R}$ is a set of rules having the form $a \rightarrow b\ c$, $a, b, c \in \mathcal{N}$, or the form $a \rightarrow x$, $a \in \mathcal{N}$ and $x \in \mathcal{L}$.

A probabilistic CFG (PCFG) is a CFG associated with a set of parameters defined as follows:

- For each $(a \rightarrow b\ c) \in \mathcal{R}$, we have a parameter $p(a \rightarrow b\ c \mid a)$.

- For each $(a \rightarrow x) \in \mathcal{R}$, we have a parameter $p(a \rightarrow x \mid a)$.
- For each $a \in \mathcal{N}$, we have a parameter π_a, which is the probability of a being the root symbol of a derivation.

The parameters above satisfy the following normalization conditions:

$$\sum_{(a \rightarrow b\ c) \in \mathcal{R}} p(a \rightarrow b\ c \mid a) + \sum_{(a \rightarrow x) \in \mathcal{R}} p(a \rightarrow x \mid a) = 1,$$

for each $a \in \mathcal{N}$, and $\sum_{a \in \mathcal{N}} \pi_a = 1$.

The probability of a tree τ deriving a sentence in the language, written $p(\tau)$, is calculated as the product of the probabilities of all rule occurrences in τ, times the parameter π_a where a is the symbol at the root of τ.

2.2 Tensor Form of PCFGs

A three-dimensional **tensor** $C \in \mathbb{R}^{(m \times m \times m)}$ is a set of m^3 parameters $C_{i,j,k}$ for $i, j, k \in [m]$. In what follows, we associate with each tensor three functions, each mapping a pair of vectors in $\mathbb{R}^m$ into a vector in $\mathbb{R}^m$.

Definition 1 *Let $C \in \mathbb{R}^{(m \times m \times m)}$ be a tensor. Given two vectors $y^1, y^2 \in \mathbb{R}^m$, we let $C(y^1, y^2)$ be the m-dimensional row vector with components:*

$$[C(y^1, y^2)]_i = \sum_{j \in [m], k \in [m]} C_{i,j,k} y^1_j y^2_k \ .$$

We also let $C_{(1,2)}(y^1, y^2)$ be the m-dimensional column vector with components:

$$[C_{(1,2)}(y^1, y^2)]_k = \sum_{i \in [m], j \in [m]} C_{i,j,k} y^1_i y^2_j \ .$$

Finally, we let $C_{(1,3)}(y^1, y^2)$ be the m-dimensional column vector with components:

$$[C_{(1,3)}(y^1, y^2)]_j = \sum_{i \in [m], k \in [m]} C_{i,j,k} y^1_i y^2_k \ .$$

For two vectors $x, y \in \mathbb{R}^m$ we denote by $x \odot y \in \mathbb{R}^m$ the Hadamard product of x and y, i.e., $[x \odot y]_i = x_i y_i$. Finally, for vectors $x, y, z \in \mathbb{R}^m$, $x y^\top z^\top$ is the tensor $D \in \mathbb{R}^{m \times m \times m}$ where $D_{i,j,k} = x_i y_j z_k$ (this is analogous to the outer product: $[x y^\top]_{i,j} = x_i y_j$).

We extend the parameter set of our PCFG such that $p(a \rightarrow b\ c \mid a) = 0$ for all $a \rightarrow b\ c$ not in $\mathcal{R}$, and $p(a \rightarrow x \mid a) = 0$ for all $a \rightarrow x$ not in $\mathcal{R}$. We also represent each $a \in \mathcal{N}$ by a unique index in $[m]$, and we represent each $x \in \mathcal{L}$ by a unique index in $[n]$: it will always be clear from the context whether these indices refer to a nonterminal in $\mathcal{N}$ or else to a word in $\mathcal{L}$.

In this paper we assume a tensor representation for the parameters $p(a \rightarrow b\ c \mid a)$, and we denote by $T \in \mathbb{R}^{m \times m \times m}$ a tensor such that:

$$T_{a,b,c} \triangleq p(a \rightarrow b\ c \mid a).$$

Similarly, we denote by $Q \in \mathbb{R}^{m \times n}$ a matrix such that:

$$Q_{a,x} \triangleq p(a \rightarrow x \mid a).$$

The root probabilities are denoted using a vector $\pi \in \mathbb{R}^{m \times 1}$ such that π_a is defined as before.

2.3 Minimum Bayes-Risk Decoding

Let $z = x_1 \cdots x_N$ be some input sentence; we write $\mathcal{T}(z)$ to denote the set of all possible trees for z. It is often the case that parsing aims to find the highest scoring tree τ^* for z according to the underlying PCFG, also called the "Viterbi parse:"

$$\tau^* = \underset{\tau \in \mathcal{T}(z)}{\text{argmax}}\ p(\tau)$$

Goodman (1996) noted that Viterbi parsers do not optimize the same metric that is usually used for parsing evaluation (Black et al., 1991). He suggested an alternative algorithm, which he called the "Labelled Recall Algorithm," which aims to fix this issue.

Goodman's algorithm has two phases. In the first phase it computes, for each $a \in \mathcal{N}$ and for each substring $x_i \cdots x_j$ of z, the marginal $\mu(a, i, j)$ defined as:

$$\mu(a, i, j) = \sum_{\tau \in \mathcal{T}(z) :\ (a,i,j) \in \tau} p(\tau).$$

Here we write $(a, i, j) \in \tau$ if nonterminal a spans words $x_i \cdots x_j$ in the parse tree τ.

Inputs: Sentence $x_1 \cdots x_N$, PCFG $(\mathcal{N}, \mathcal{L}, \mathcal{R})$, parameters $T \in \mathbb{R}^{(m \times m \times m)}$, $Q \in \mathbb{R}^{(m \times n)}$, $\pi \in \mathbb{R}^{(m \times 1)}$.

Data structures:

- Each $\mu(a, i, j) \in \mathbb{R}$ for $a \in \mathcal{N}$, $i, j \in [N]$, $i \leq j$, is a marginal probability.
- Each $\gamma^{i,j} \in \mathbb{R}$ for $i, j \in [N]$, $i \leq j$, is the highest score for a tree spanning substring $x_i \cdots x_j$.

Algorithm:

(Marginals) $\forall a \in \mathcal{N}, \forall i, j \in [N], i \leq j$, compute the marginals $\mu(a, i, j)$ using the inside-outside algorithm.

(Base case) $\forall i \in [N]$,

$$\gamma^{i,i} = \max_{(a \to x_i) \in \mathcal{R}} \mu(a, i, i)$$

(Maximize Labelled Recall) $\forall i, j \in [N], i < j$,

$$\gamma^{i,j} = \max_{a \in \mathcal{N}} \mu(a, i, j) + \max_{i \leq k < j} \left(\gamma^{i,k} + \gamma^{k+1,j} \right)$$

Figure 1: The labelled recall algorithm from Goodman (1996). The algorithm in this figure finds the highest score for a tree which maximizes labelled recall. The actual parsing algorithm would use backtrack pointers in the score computation to return a tree. These are omitted for simplicity.

The second phase includes a dynamic programming algorithm which finds the tree τ^* that maximizes the sum over marginals in that tree:

$$\tau^* = \operatorname*{argmax}_{\tau \in \mathcal{T}(z)} \sum_{(a,i,j) \in \tau} \mu(a, i, j).$$

Goodman's algorithm is described in Figure 1.

As Goodman notes, the complexity of the second phase ("Maximize Labelled Recall," which is also referred to as "minimum Bayes risk decoding") is $\mathcal{O}(N^3 + mN^2)$. There are two nested outer loops, each of order N, and inside these, there are two separate loops, one of order m and one of order N, yielding this computational complexity. The reason for the linear dependence on the number of nonterminals is the lack of dependence on the actual grammar rules, once the marginals are computed.

In its original form, Goodman's algorithm does not enforce that the output parse trees are included in the tree language of the PCFG, that is, certain combinations of children and parent nonterminals may violate the rules in the grammar. In our experiments we departed from this, and changed Goodman's algorithm by incorporating the grammar into the dynamic programming algorithm in Figure 1. The reason this is important for our experiments is that we *binarize* the grammar prior to parsing, and we need to enforce the links between the split nonterminals (in the binarized grammar) that refer to the same syntactic category. See Matsuzaki et al. (2005) for more details about the binarization scheme we used. This step changes the dynamic programming equation of Goodman to be linear in the size of the grammar (figure 1). However, empirically, it is the inside-outside algorithm which takes most of the time to compute with Goodman's algorithm. In this paper we aim to asymptotically reduce the time complexity of the calculation of the inside-outside probabilities using an approximation algorithm.

3 Tensor Formulation of the Inside-Outside Algorithm

At the core of our approach lies the observation that there is a (multi)linear algebraic formulation of the inside-outside algorithm. It can be represented as a series of tensor, matrix and vector products. A similar observation has been made for latent-variable PCFGs (Cohen et al., 2012) and hidden Markov models, where only matrix multiplication is required (Jaeger, 2000). Cohen and Collins (2012) use this observation together with tensor decomposition to improve the speed of latent-variable PCFG parsing.

The representation of the inside-outside algorithm in tensor form is given in Figure 2. For example, if we consider the recursive equation for the inside probabilities (where $\alpha^{i,j}$ is a vector varying over the nonterminals in the grammar, describing the inside probability for each nonterminal spanning words i to j):

$$\alpha^{i,j} = \sum_{k=i}^{j-1} T(\alpha^{i,k}, \alpha^{k+1,j})$$

Inputs: Sentence $x_1 \cdots x_N$, PCFG $(\mathcal{N}, \mathcal{L}, \mathcal{R})$, parameters $T \in \mathbb{R}^{(m \times m \times m)}$, $Q \in \mathbb{R}^{(m \times n)}$, $\pi \in \mathbb{R}^{(m \times 1)}$.

Data structures:

- Each $\alpha^{i,j} \in \mathbb{R}^{1 \times m}$, $i, j \in [N]$, $i \leq j$, is a row vector of inside terms ranging over $a \in \mathcal{N}$.
- Each $\beta^{i,j} \in \mathbb{R}^{m \times 1}$, $i, j \in [N]$, $i \leq j$, is a column vector of outside terms ranging over $a \in \mathcal{N}$.
- Each $\mu(a, i, j) \in \mathbb{R}$ for $a \in \mathcal{N}$, $i, j \in [N]$, $i \leq j$, is a marginal probability.

Algorithm:

(Inside base case) $\forall i \in [N], \forall (a \rightarrow x_i) \in \mathcal{R}$,

$$[\alpha^{i,i}]_a = Q_{a,x}$$

(Inside recursion) $\forall i, j \in [N], i < j$,

$$\alpha^{i,j} = \sum_{k=i}^{j-1} T(\alpha^{i,k}, \alpha^{k+1,j})$$

(Outside base case) $\forall a \in \mathcal{N}$,

$$[\beta^{1,N}]_a = \pi_a$$

(Outside recursion) $\forall i, j \in [N], i \leq j$,

$$\beta^{i,j} = \sum_{k=1}^{i-1} T_{(1,2)}(\beta^{k,j}, \alpha^{k,i-1}) + \sum_{k=j+1}^{N} T_{(1,3)}(\beta^{i,k}, \alpha^{j+1,k})$$

(Marginals) $\forall a \in \mathcal{N}, \forall i, j \in [N], i \leq j$,

$$\mu(a, i, j) = [\alpha^{i,j}]_a \cdot [\beta^{i,j}]_a$$

Figure 2: The tensor form of the inside-outside algorithm, for calculation of marginal terms $\mu(a, i, j)$.

and then apply the tensor product from Definition 1 to this equation, we get that coordinate a in $\alpha^{i,j}$ is defined recursively as follows:

$$[\alpha^{i,j}]_a = \sum_{k=i}^{j-1} \sum_{b,c} T_{a,b,c} \times \alpha_b^{i,k} \times \alpha_c^{k+1,j}$$
$$= \sum_{k=i}^{j-1} \sum_{b,c} p(a \rightarrow b\ c \mid a) \times \alpha_b^{i,k} \times \alpha_c^{k+1,j},$$

which is exactly the recursive definition of the inside algorithm. The correctness of the outside recursive equations follows very similarly.

The time complexity of the algorithm in this case is $\mathcal{O}(m^3 N^3)$. To see this, observe that each tensor application takes time $\mathcal{O}(m^3)$. Furthermore, the tensor T is applied $\mathcal{O}(N)$ times in the computation of each vector $\alpha^{i,j}$ and $\beta^{i,j}$. Finally, we need to compute a total of $\mathcal{O}(N^2)$ inside and outside vectors, one for each substring of the input sentence.

4 Tensor Decomposition for the Inside-Outside Algorithm

In this section, we detail our approach to approximate parsing using tensor decomposition.

4.1 Tensor Decomposition

In the formulation of the inside-outside algorithm based on tensor T, each vector $\alpha^{i,j}$ and $\beta^{i,j}$ consists of m elements, where computation of each element requires time $\mathcal{O}(m^2)$. Therefore, the algorithm has a $\mathcal{O}(m^3)$ multiplicative factor in its time complexity, which we aim to reduce by means of an approximate algorithm.

Our approximate method relies on a simple observation. Given an integer $r \geq 1$, assume that the tensor T has the following special form, called "Kruskal form:"

$$T = \sum_{i=1}^{r} \lambda_i u_i v_i^\top w_i^\top \ . \tag{1}$$

In words, T is the sum of r tensors, where each tensor is obtained as the product of three vectors u_i, v_i and w_i, together with a scalar λ_i. Exact Kruskal decomposition of a tensor is not necessarily unique. See Kolda and Bader (2009) for discussion of uniqueness of tensor decomposition.

Consider now two vectors $y^1, y^2 \in \mathbb{R}^m$, associated with the inside probabilities for the left (y^1) and right child (y^2) of a given node in a parse tree. Let us introduce auxiliary arrays $U, V, W \in \mathbb{R}^{r \times m}$, with the i-th row being u_i, v_i and w_i, respectively. Let also $\lambda = (\lambda_1, \ldots, \lambda_r)$. Using the decomposition in Eq. 1 within Definition 1 we can express the array $T(y^1, y^2)$ as:

$$T(y^1, y^2) = \left[\sum_{i=1}^{r} \lambda_i u_i v_i^\top w_i^\top\right](y^1, y^2) = \sum_{i=1}^{r} \lambda_i u_i (v_i^\top y^1)(w_i^\top y^2) = \left(U^\top(\lambda \odot V y^1 \odot W y^2)\right) . \quad (2)$$

The total complexity of the computation in Eq. 2 is now $\mathcal{O}(rm)$. It is well-known that an exact tensor decomposition for T can be achieved with $r = m^2$ (Kruskal, 1989). In this case, there is no computational gain in using Eq. 2 for the inside calculation. The minimal r required for an exact tensor decomposition can be smaller than m^2. However, identifying that minimal r is NP-hard (Høastad, 1990).

In this section we focused on the computation of the inside probabilities through vectors $T(\alpha^{i,k}, \alpha^{k+1,j})$. Nonetheless, the steps above can be easily adapted for the computation of the outside probabilities through vectors $T_{(1,2)}(\beta^{k,j}, \alpha^{k,i-1})$ and $T_{(1,3)}(\beta^{i,k}, \alpha^{j+1,k})$.

4.2 Approximate Tensor Decomposition

The PCFG tensor T will not necessarily have the exact decomposed form in Eq. 1. We suggest to *approximate* the tensor T by finding the closest tensor according to some norm over $\mathbb{R}^{m \times m \times m}$.

An example of such an approximate decomposition is the canonical polyadic decomposition (CPD), also known as CANDECOMP/PARAFAC decomposition (Carroll and Chang, 1970; Harshman, 1970; Kolda and Bader, 2009). Given an integer r, *least squares* CPD aims to find the nearest tensor in Kruskal form, minimizing squared error.

More formally, for a given tensor $D \in \mathbb{R}^{m \times m \times m}$, let $\|D\|_F = \sqrt{\sum_{i,j,k} D_{i,j,k}^2}$. Let the set of tensors in Kruskal form $\mathcal{C}_r$ be:

$$\mathcal{C}_r = \{C \in \mathbb{R}^{m \times m \times m} \mid C = \sum_{i=1}^{r} \lambda_i u_i v_i^\top w_i^\top \text{ s.t. } \lambda_i \in \mathbb{R}, u_i, v_i, w_i \in \mathbb{R}^m, \|u_i\|_2 = \|v_i\|_2 = \|w_i\|_2 = 1\}.$$

The least squares CPD of C is a tensor $\hat{C}$ such that $\hat{C} \in \operatorname{argmin}_{\hat{C} \in \mathcal{C}_r} \|C - \hat{C}\|_F$. Here, we treat the argmin as a set because there could be multiple solutions which achieve the same accuracy.

There are various algorithms to perform CPD, such as alternating least squares, direct linear decomposition, alternating trilinear decomposition and pseudo alternating least squares (Faber et al., 2003) and even algorithms designed for sparse tensors (Chi and Kolda, 2011). Most of these algorithms treat the problem of identifying the approximate tensor as an optimization problem. Generally speaking, these optimization problems are hard to solve, but they work quite well in practice.

4.3 Parsing with Decomposed Tensors

Equipped with the notion of tensor decomposition, we can now proceed with approximate tensor parsing in two steps. The first is approximating the tensor using a CPD algorithm, and the second is applying the algorithms in Figure 1 and Figure 2 to do parsing, while substituting all tensor product computations with the approximate $\mathcal{O}(rm)$ operation of tensor product.

This is not sufficient to get a significant speed-up in parsing time. Re-visiting Eq. 2 shows that there are additional ways to speed-up the tensor application T in the context of the inside-outside algorithm.

The first thing to note is that the projections Vy^1 and Wy^2 in Eq. 2 can be cached, and do not have to be re-calculated every time the tensor is applied. Here, y^1 and y^2 will always refer to an outside or an inside probability vector over the nonterminals in the grammar. Caching these projections means that after each computation of an inside or outside probability, we can immediately project it to the necessary r-dimensional space, and then re-use this computation in subsequent application of the tensor.

The second thing to note is that the U projection in T can be delayed, because of the rule of

distributivity. For example, the step in Figure 2 that computes the inside probability $\alpha^{i,j}$ can be reformulated as follows (assuming an exact decomposition of T):

$$\begin{aligned}\alpha^{i,j} &= \sum_{k=i}^{j-1} T(\alpha^{i,k}, \alpha^{k+1,j}) \\ &= \sum_{k=1}^{j-1} U^\top(\lambda \odot V\alpha^{i,k} \odot W\alpha^{k+1,j}) \\ &= U^\top \left(\sum_{k=1}^{j-1} (\lambda \odot V\alpha^{i,k} \odot W\alpha^{k+1,j}) \right). \quad (3)\end{aligned}$$

This means that projection through U can be done outside of the loop over splitting points in the sentence. Similar reliance on distributivity can be used to speed-up the outside calculations as well.

The caching speed-up and the delayed projection speed-up make the approximate inside-outside computation asymptotically faster. While naïve application of the tensor yields an inside algorithm which runs in time $\mathcal{O}(rmN^3)$, the improved algorithm runs in time $\mathcal{O}(rN^3 + rmN^2)$.

5 Quality of Approximate Tensor Parsing

In this section, we give the main approximation result, that shows that the probability distribution induced by the approximate tensor is close to the original probability distribution, if the distance between the approximate tensor and the rule probabilities is not too large.

Denote by $\mathcal{T}(N)$ the set of trees in the tree language of the PCFG with N words (any nonterminal can be the root of the tree). Let $\overline{\mathcal{T}}(N)$ be the set of pairs of trees $\overline{\tau} = (\tau_1, \tau_2)$ such that the total number of binary rules combined in τ_1 and τ_2 is $N-2$ (this means that the total number of words combined is N). Let $\hat{T}$ be the approximate tensor for T. Denote the probability distribution induced by $\hat{T}$ by $\hat{p}$. [1] Define the vector $\xi(\overline{\tau})$ such that $[\xi(\overline{\tau})]_a = T_{a,b,c} \cdot p(\tau_1 \mid b) \cdot p(\tau_2 \mid c)$ where the root τ_1 is nonterminal b and the root of τ_2 is c. Similarly, define $[\hat{\xi}(\overline{\tau})]_a = \hat{T}_{a,b,c} \cdot \hat{p}(\tau_1 \mid b) \cdot \hat{p}(\tau_2 \mid c)$.

[1] Here, $\hat{p}$ does not have to be a distribution, because $\hat{T}$ could have negative values, in principle, and its slices do not have to normalize to 1. However, we just treat $\hat{p}$ as a function that maps trees to products of values according to $\hat{T}$.

Define $Z(a,N) = \sum_{\overline{\tau} \in \overline{\mathcal{T}}(N)} [\hat{\xi}(\overline{\tau})]_a$. In addition, define $D(a,N) = \sum_{\overline{\tau} \in \overline{\mathcal{T}}(N)} |[\hat{\xi}(\overline{\tau})]_a - [\xi(\overline{\tau})]_a|$ and define $F(a,N) = D(a,N)/Z(a,N)$. Define $\Delta = \|\hat{T} - T\|_F$. Last, define $\nu = \min_{(a \to b\ c) \in \mathcal{R}} p(a \to b\ c \mid a)$. Then, the following lemma holds:

Lemma 1 *For any a and any N, it holds:*

$$D(a,N) \leq Z(a,N) \left((1 + \Delta/\nu)^N - 1 \right)$$

Proof sketch: The proof is by induction on N. Assuming that $1 + F(b,k) \leq (1 + \Delta/\nu)^k$ and $1 + F(c, N-k-1) \leq (1 + \Delta/\nu)^{N-k-1}$ for F defined as above (this is the induction hypothesis), it can be shown that the lemma holds. ■

Lemma 2 *The following holds for any N:*

$$\sum_{\tau \in \mathcal{T}(N)} |\hat{p}(\tau) - p(\tau)| \leq m \left((1 + \Delta/\nu)^N - 1 \right)$$

Proof sketch: Using Hölder's inequality and Lemma 1 and the fact that $Z(a,N) \leq 1$, it follows that:

$$\begin{aligned}\sum_{\tau \in \mathcal{T}(N)} |\hat{p}(\tau) - p(\tau)| &\leq \sum_{\overline{\tau} \in \overline{\mathcal{T}}(N), a} |[\xi(\overline{\tau})]_a - [\hat{\xi}(\overline{\tau})]_a| \\ &\leq \left(\sum_a Z(a,N) \right) \left((1 + \Delta/\nu)^N - 1 \right) \\ &\leq m \left((1 + \Delta/\nu)^N - 1 \right)\end{aligned}$$

■

Then, the following is a result that explains how accuracy changes as a function of the quality of the tensor approximation:

Theorem 1 *For any N, and $\epsilon < 1/4$, it holds that if $\Delta \leq \frac{\epsilon\nu}{2Nm}$, then:*

$$\sum_{\tau \in \mathcal{T}(N)} |\hat{p}(\tau) - p(\tau)| \leq \epsilon \quad (4)$$

Proof sketch: This is the result of applying Lemma 2 together with the inequality $(1 + y/t)^t - 1 \leq 2y$ for any $t > 0$ and $y \leq 1/2$. ■

We note that Theorem 1 also implicitly bounds the difference between a marginal $\mu(a, i, j)$ and its approximate version. A marginal corresponds to a sum over a subset of summands in Eq. 4.

A question that remains at this point is to decide whether for a given grammar, the optimal ν that can be achieved is large or small. We define:

$$\Delta_r^* = \min_{\hat{T} \in \mathcal{C}_r} \|T - \hat{T}\|_F \tag{5}$$

The following theorem gives an upper bound on the value of Δ_r^* based on an intrinsic property of the grammar, or more specifically T. It relies on the fact that for three-dimensional tensors, where each dimension is of length m, there exists an exact decomposition of T using m^2 components.

Theorem 2 *Let:*

$$T = \sum_{i=1}^{m^2} \lambda_i^* u_i^* (v_i^*)^\top (w_i^*)^\top$$

be an exact Kruskal decomposition of T such that $\|u_i^*\|_2 = \|v_i^*\|_2 = \|w_i^*\| = 1$ and $\lambda_i^* \geq \lambda_{i+1}^*$ for $i \in [m^2 - 1]$. Then, for a given r, it holds:

$$\Delta_r^* \leq \sum_{i=r+1}^{m^2} |\lambda_i^*|$$

Proof: Let $\hat{T}$ be a tensor that achieves the minimum in Eq. 5. Define:

$$T'_r = \sum_{i=1}^{r} \lambda_i^* u_i^* (v_i^*)^\top (w_i^*)^\top$$

Then, noting that Δ_r^* is a minimizer of the norm difference between T and $\hat{T}$ and then applying the triangle inequality and then Cauchy-Schwartz inequality leads to the following chain of inequalities:

$$\begin{aligned}
\Delta_r^* &= \|T - \hat{T}\|_F \leq \|T - T'_r\|_F \\
&= \| \sum_{i=r+1}^{m^2} \lambda_i^* u_i^* (v_i^*)^\top (w_i^*)^\top \|_F \\
&\leq \sum_{i=r+1}^{m^2} |\lambda_i^*| \cdot \|u_i^* (v_i^*)^\top (w_i^*)^\top\|_F \quad = \sum_{i=r+1}^{m^2} |\lambda_i^*|
\end{aligned}$$

as required. ■

6 Experiments

In this section, we describe experiments that demonstrate the trade-off between the accuracy of the tensor approximation (and as a consequence, the accuracy of the approximate parsing algorithm) and parsing time.

Experimental Setting We compare the tensor approximation parsing algorithm versus the vanilla Goodman algorithm. Both algorithms were implemented in Java, and the code for both is almost identical, except for the set of instructions which computes the dynamic programming equation for propagating the beliefs up in the tree. This makes the clocktime comparison reliable for drawing conclusions about the speed of the algorithms. Our implementation of the vanilla parsing algorithm is linear in the size of the grammar (and not cubic in the number of nonterminals, which would give a worse running time).

In our experiments, we use the method described in Chi and Kolda (2011) for tensor decomposition.[2] This method is fast, even for large tensors, as long as they are sparse. Such is the case with the tensors for our grammars.

We use two treebanks for our comparison: the Penn treebank (Marcus et al., 1993) and the Arabic treebank (Maamouri et al., 2004). With the Penn treebank, we use sections 2–21 for training a maximum likelihood model and section 22 for parsing, while for the Arabic treebank we divide the data into

[2] We use the implementation given in Sandia's Matlab Tensor Toolbox, which can be downloaded at `http://www.sandia.gov/~tgkolda/TensorToolbox/index-2.5.html`.

two sets, of size 80% and 20%, one is used for training a maximum likelihood model and the other is used for parsing.

The number of binary rules in the treebank grammar is 7,240. The number of nonterminals is 112 and the number of preterminals is 259.[3] The number of binary rules in the Arabic treebank is significantly smaller and consists of 232 rules. We run all parsing experiments on sentences of length ≤ 40. The number of nonterminals is 48 and the number of preterminals is 81.

Results Table 1 describes the results of comparing the tensor decomposition algorithm to the vanilla PCFG parsing algorithm.

The first thing to note is that the running time of the parsing algorithm is linear in r. This indeed validates the asymptotic complexity of the inside-outside component in Goodman's algorithm with the approximate tensors. It also shows that most of the time during parsing is spent on the inside-outside algorithm, and not on the dynamic programming algorithm which follows it.

In addition, compared to the baseline which uses a vanilla CKY algorithm (linear in the number of rules), we get a speed up of a factor of 4.75 for Arabic ($r = 140$) and 6.5 for English ($r = 260$) while retaining similar performance. Perhaps more surprising is that using the tensor approximation actually *improves* performance in several cases. We hypothesize that the cause of this is that the tensor decomposition requires less parameters to express the rule probabilities in the grammar, and therefore leads to better generalization than a vanilla maximum likelihood estimate.

We include results for a more complex model for Arabic, which uses horizontal Markovization of order 1 and vertical Markovization of order 2 (Klein and Manning, 2003). This grammar includes 2,188 binary rules. Parsing exhaustively using this grammar takes 1.30 seconds per sentence (on average) with an F_1 measure of 64.43. Parsing with tensor decomposition for $r = 280$ takes 0.62 seconds per sentence (on average) with an F_1 measure of 64.05.

[3]Unary rules are removed by collapsing non-terminal chains. This increased the number of preterminals.

7 Discussion

In this section, we briefly touch on several other topics related to tensor approximation.

7.1 Approximating the Probability of a String

The probability of a sentence z under a PCFG is defined as $p(z) = \sum_{\tau \in \mathcal{T}(z)} p(\tau)$, and can be approximated using the algorithm in Section 4.3, running in time $\mathcal{O}(rN^3 + rmN^2)$. Of theoretical interest, we discuss here a time $\mathcal{O}(rN^3 + r^2N^2)$ algorithm, which is more convenient when $r < m$.

Observe that in Eq. 3 vector $\alpha^{i,j}$ always appears within one of the two terms $V\alpha^{i,j}$ and $W\alpha^{i,j}$ in $\mathbb{R}^{r \times 1}$, whose dimensions are independent of m. We can therefore use Eq. 3 to compute $V\alpha^{i,j}$ as

$$V\alpha^{i,j} = VU^\top \left(\sum_{k=1}^{j-1} (\lambda \odot V\alpha^{i,k} \odot W\alpha^{k+1,j}) \right),$$

where $VU^\top$ is a $\mathbb{R}^{r \times r}$ matrix that can be computed off-line, i.e., independently of z. A symmetrical relation can be used to compute $W\alpha^{i,j}$. Finally, we can write

$$p(z) = \pi^\top U \left(\sum_{k=1}^{N-1} (\lambda \odot V\alpha^{1,k} \odot W\alpha^{k+1,N}) \right),$$

where $\pi^\top U$ is a $\mathbb{R}^{1 \times r}$ vector that can again be computed off-line. This algorithm then runs in time $\mathcal{O}(rN^3 + r^2N^2)$.

7.2 Applications to Dynamic Programming

The approximation method presented in this paper is not limited to PCFG parsing. A similar approximation method has been used for latent-variable PCFGs (Cohen and Collins, 2012), and in general, tensor approximation can be used to speed-up inside-outside algorithms for general dynamic programming algorithms or weighted logic programs (Eisner et al., 2004; Cohen et al., 2011). In the general case, the dimension of the tensors will not be necessarily just three (corresponding to binary rules), but can be of a higher dimension, and therefore the speed gain can be even greater. In addition, tensor approximation can be used for computing marginals of latent variables in graphical models.

For example, the complexity of the forward-backward algorithm for HMMs can be reduced to

	rank (r)	baseline	20	60	100	140	180	220	260	300	340
Arabic	speed	0.57	0.04	0.06	0.1	0.12	0.16	0.19	0.22	0.26	0.28
	F_1	63.78	51.80	58.39	63.63	63.77	63.88	63.82	63.84	63.80	63.88
English	speed	3.89	0.15	0.21	0.30	0.37	0.44	0.52	0.60	0.70	0.79
	F_1	71.07	57.83	61.67	68.28	69.63	70.30	70.82	71.42	71.28	71.13

Table 1: Results for the Arabic and English treebank of parsing using a vanilla PCFG with and without tensor decomposition. Speed is given in seconds per sentence.

be linear in the number of states (as opposed to quadratic) and linear in the rank used in an approximate singular-value decomposition (instead of tensor decomposition) of the transition and emission matrices.

7.3 Tighter (but Slower) Approximation Using Singular Value Decomposition

The accuracy of the algorithm depends on the ability of the tensor decomposition algorithm to decompose the tensor with a small reconstruction error. The decomposition algorithm is performed on the tensor T which includes all rules in the grammar.

Instead, one can approach the approximation by doing a decomposition for each *slice* of T separately using singular value decomposition. This will lead to a more accurate approximation, but will also lead to an extra factor of m during parsing. This factor is added because now there is not a single U, V and W, but instead there are such matrices for each nonterminal in the grammar.

8 Conclusion

We described an approximation algorithm for probabilistic context-free parsing. The approximation algorithm is based on tensor decomposition performed on the underlying rule table of the CFG grammar. The approximation algorithm leads to significant speed-up in PCFG parsing, with minimal loss in performance.

References

E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of DARPA Workshop on Speech and Natural Language*.

P. Boullier. 2003. Guided earley parsing. In *8th International Workshop on Parsing Technologies*, pages 43–54.

J. D. Carroll and J. J. Chang. 1970. Analysis of individual differences in multidimensional scaling via an N-way generalization of Eckart-Young decomposition. *Psychometrika*, 35:283–319.

E. Charniak, M. Johnson, M. Elsner, J. Austerweil, D. Ellis, I. Haxton, C. Hill, R. Shrivaths, J. Moore, M. Pozar, and T. Vu. 2006. Multilevel coarse-to-fine pcfg parsing. In *Proceedings of HLT-NAACL*.

E. C. Chi and T. G. Kolda. 2011. On tensors, sparsity, and nonnegative factorizations. arXiv:1112.2414 [math.NA], December.

S. B. Cohen and M. Collins. 2012. Tensor decomposition for fast parsing with latent-variable PCFGs. In *Proceedings of NIPS*.

S. B. Cohen, R. J. Simmons, and N. A. Smith. 2011. Products of weighted logic programs. *Theory and Practice of Logic Programming*, 11(2–3):263–296.

S. B. Cohen, K. Stratos, M. Collins, D. F. Foster, and L. Ungar. 2012. Spectral learning of latent-variable PCFGs. In *Proceedings of ACL*.

J. Eisner and N. A. Smith. 2005. Parsing with soft and hard constraints on dependency length. In *Proceedings of IWPT*, Parsing '05.

J. Eisner, E. Goldlust, and N. A. Smith. 2004. Dyna: A declarative language for implementing dynamic programs. In *Proc. of ACL* (companion volume).

N. M. Faber, R. Bro, and P. Hopke. 2003. Recent developments in CANDECOMP/PARAFAC algorithms: a critical review. *Chemometrics and Intelligent Laboratory Systems*, 65(1):119–137.

J. Goodman. 1996. Parsing algorithms and metrics. In *Proceedings of ACL*.

R. A. Harshman. 1970. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA working papers in phoentics*, 16:1–84.

J. Høastad. 1990. Tensor rank is NP-complete. *Algorithms*, 11:644–654.

H. Jaeger. 2000. Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398.

D. Klein and C. D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*.

T. G. Kolda and B. W. Bader. 2009. Tensor decompositions and applications. *SIAM Rev.*, 51:455–500.

J. B. Kruskal. 1989. Rank, decomposition, and uniqueness for 3-way and N-way arrays. In R. Coppi and S. Bolasco, editors, *Multiway Data Analysis*, pages 7–18.

M. Maamouri, A. Bies, T. Buckwalter, and W. Mekki. 2004. The Penn Arabic Treebank: Building a large-scale annotated Arabic corpus. In *Proceedings NEMLAR*.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of ACL*.

M.-J. Nederhof. 2000. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1):17–44.