Sampling Tree Fragments from Forests

Tagyoung Chung*
University of Rochester

Licheng Fang**
University of Rochester

Daniel Gildea†
University of Rochester

Daniel Štefankovič‡
University of Rochester

We study the problem of sampling trees from forests, in the setting where probabilities for each tree may be a function of arbitrarily large tree fragments. This setting extends recent work for sampling to learn Tree Substitution Grammars to the case where the tree structure (TSG derived tree) is not fixed. We develop a Markov chain Monte Carlo algorithm which corrects for the bias introduced by unbalanced forests, and we present experiments using the algorithm to learn Synchronous Context-Free Grammar rules for machine translation. In this application, the forests being sampled represent the set of Hiero-style rules that are consistent with fixed input word-level alignments. We demonstrate equivalent machine translation performance to standard techniques but with much smaller grammars.

1. Introduction

Recent work on learning Tree Substitution Grammars (TSGs) has developed procedures for sampling TSG rules from known derived trees (Cohn, Goldwater, and Blunsom 2009; Post and Gildea 2009). Here one samples binary variables at each node in the tree, indicating whether the node is internal to a TSG rule or is a split point between two rules. We consider the problem of learning TSGs in cases where the tree structure is not known, but rather where possible tree structures are represented in a forest. For example, we may wish to learn from text where treebank annotation is unavailable,

* Computer Science Dept., University of Rochester, Rochester NY 14627.
E-mail: chung@cs.rochester.edu.
** Computer Science Dept., University of Rochester, Rochester NY 14627.
E-mail: lfang@cs.rochester.edu.
† Computer Science Dept., University of Rochester, Rochester NY 14627.
E-mail: gildea@cs.rochester.edu.
‡ Computer Science Dept., University of Rochester, Rochester NY 14627.
E-mail: stefanko@cs.rochester.edu.

Submission received: 26 October 2012; revised version received: 14 March 2013; accepted for publication: 4 May 2013.

doi:10.1162/COLI_a_00170

but a forest of likely parses can be produced automatically. Another application on which we focus our attention in this article arises in machine translation, where we want to learn translation rules from a forest representing the phrase decompositions that are consistent with an automatically derived word alignment. Both these applications involve sampling TSG trees from forests, rather than from fixed derived trees.

Chappelier and Rajman (2000) present a widely used algorithm for sampling trees from forests: One first computes an inside probability for each node bottom–up, and then chooses an incoming hyperedge for each node top–down, sampling according to each hyperedge's inside probability. Johnson, Griffiths, and Goldwater (2007) use this sampling algorithm in a Markov chain Monte Carlo framework for grammar learning. We can combine the representations used in this algorithm and in the TSG learning algorithm discussed earlier, maintaining two variables at each node of the forest, one for the identity of the incoming hyperedge, and another representing whether the node is internal to a TSG rule or is a split point. However, computing an inside probability for each node, as in the first phase of the algorithm of Johnson, Griffiths, and Goldwater (2007), becomes difficult because of the exponential number of TSG rules that can apply at any node in the forest. Not only is the number of possible TSG rules that can apply given a fixed tree structure exponentially large in the size of the tree, but the number of possible tree structures under a node is also exponentially large. This problem is particularly acute during grammar learning, as opposed to sampling according to a fixed grammar, because any tree fragment is a valid potential rule. Cohn and Blunsom (2010) address the large number of valid unseen rules by decomposing the prior over TSG rules into an equivalent probabilistic context-free grammar; however, this technique only applies to certain priors. In general, algorithms that match all possible rules are likely to be prohibitively slow, as well as unwieldy to implement. In this article, we design a sampling algorithm that avoids explicitly computing inside probabilities for each node in the forest.

In Section 2, we derive a general algorithm for sampling tree fragments from forests. We avoid computing inside probabilities, as in the TSG sampling algorithms of Cohn, Goldwater, and Blunsom (2009) and Post and Gildea (2009), but we must correct for the bias introduced by the forest structure, a complication that does not arise when the tree structure is fixed. In order to simplify the presentation of the algorithm, we first set aside the complication of large, TSG-style rules, and describe an algorithm for sampling trees from forests while avoiding computation of inside probabilities. This algorithm is then generalized to learn the composed rules of TSG in Section 2.3.

As an application of our technique, we present machine translation experiments in the remainder of the article. We learn Hiero-style Synchronous Context-Free Grammar (SCFG) rules (Chiang 2007) from bilingual sentences for which a forest of possible minimal SCFG rules has been constructed from fixed word alignments. The construction of this forest and its properties are described in Section 3. We make the assumption that the alignments produced by a word-level model are correct in order to simplify the computation necessary for rule learning. This approach seems safe given that the pipeline of alignment followed by rule extraction has generally remained the state of the art despite attempts to learn joint models of alignment and rule decomposition (DeNero, Bouchard-Cote, and Klein 2008; Blunsom et al. 2009; Blunsom and Cohn 2010a). We apply our sampling algorithm to learn the granularity of rule decomposition in a Bayesian framework, comparing sampling algorithms in Section 4. The end-to-end machine translation experiments of Section 5 show that our algorithm is able to achieve performance equivalent to the standard technique of extracting all rules, but results in a significantly smaller grammar.

2. Sampling Trees from Forests

As a motivating example, consider the small example forest of Figure 1. This forest contains a total of five trees, one under the hyperedge labeled A, and four under the hyperedge labeled B (the cross-product of the two options for deriving node 4 and the two options for deriving node 5).

Let us suppose that we wish to sample trees from this forest according to a distribution P_t, and further suppose that this distribution is proportional to the product of the weights of each tree's hyperedges:

$$P_t(t) \propto \prod_{h \in t} w(h) \tag{1}$$

To simplify the example, suppose that in Figure 1 each hyperedge has weight 1,

$$\forall h \qquad w(h) = 1$$

giving us a uniform distribution over trees:

$$\forall t \qquad P_t(t) = \frac{1}{5}$$

A tree can be specified by attaching a variable z_n to each node n in the forest indicating which incoming hyperedge is to be used in the current tree. For example, variable z_1 can take values A and B in Figure 1, whereas variables z_2 and z_3 can only take a single value. We use z to refer to the entire set of variables z_n in a forest. Each assignment to z specifies a unique tree, $\tau(z)$, which can be found by following the incoming hyperedges specified by z from the goal node of each forest down to the terminals.

A naive sampling strategy would be to resample each of these variables z_n in order, holding all others constant, as in standard Gibbs sampling. If we choose an incoming hyperedge according to the probability $P_t(\tau(z))$ of the resulting tree, holding all other variable assignments fixed, we see that, because P_t is uniform, we will choose

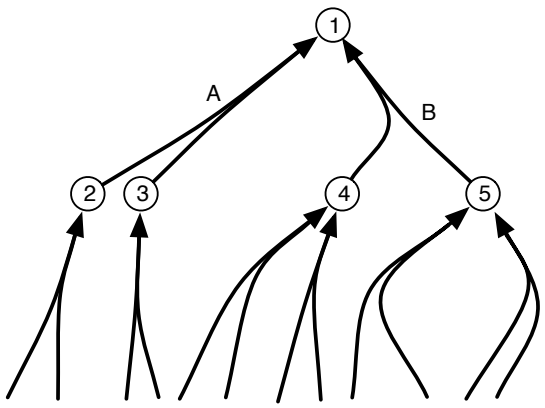

Figure 1
Example forest.

with uniform probability of $1/m$ among the m incoming hyperedges at each node. In particular, we will choose among the two incoming hyperedges at the root (node 1) with equal probability, meaning that, over the long run, the sampler will spend half its time in the state for the single tree corresponding to nodes 2 and 3, and only one eighth of its time in each of the four other possible trees. Our naive algorithm has failed at its goal of sampling among the five possible trees each with probability 1/5.

Thus, we cannot adopt the simple Gibbs sampling strategy, used for TSG induction from fixed trees, of resampling one variable at a time according to the target distribution, conditioned on all other variables. The intuitive reason for this, as illustrated by the example, is the bias introduced by forests that are bushier (that is, have more derivations for each node) in some parts than in others. The algorithm derived in the remainder of this section corrects for this bias, while avoiding the computation of inside probabilities in the forest.

2.1 Choosing a Stationary Distribution

We will design our sampling algorithm by first choosing a distribution P_z over the set of variables z defined earlier. We will show correctness of our algorithm by showing that it is a Markov chain converging to P_z, and that P_z results in the desired distribution P_t over trees.

The tree specified by an assignment to z will be denoted $\tau(z)$ (see Table 1). For a tree t the vector containing the variables found at nodes in t will be denoted $z[t]$. This is a subvector of z: for example, in Figure 1, if t chooses A at node 1 then $z[t] = (z_1, z_2, z_3)$, and if t chooses B at node 1 then $z[t] = (z_1, z_4, z_5)$. We use $z[\neg t]$ to denote the variables not used in the tree t. The vectors $z[t]$ and $z[\neg t]$ differ according to t, but for any tree t, the two vectors form a partition of z. There are many values of z that correspond to the same tree, but each tree t corresponds to a unique subvector of variables $z[t]$ and a unique assignment to those specific variables—we will denote this unique assignment of variables in $z[t]$ by $\zeta(t)$. (In terms of τ and ζ one has for any z and t that $\tau(z) = t$ if and only if $z[t] = \zeta(t)$.)

Let Z be the random vector generated by our algorithm. As long as $P_z(\tau(Z) = t) = P_t(t)$ for all trees t, our algorithm will generate trees from the desired distribution. Thus for any tree t the probability $P_z(Z[t] = \zeta(t))$ is fixed to $P_t(t)$, but any distribution over the remaining variables not contained in t will still yield the desired distribution over trees. Thus, in designing the sampling algorithm, we may choose any distribution $P_z(Z[\neg t] \mid Z[t] = \zeta(t))$. A simple and convenient choice is to make $P_z(Z[\neg t] \mid Z[t] = \zeta(t))$ uniform. That is, each incoming hyperedge variable with m alternatives assigns each hyperedge probability $1/m$. (In fact, our algorithm can easily

Table 1
Notation

P_t	desired distribution on trees
z	vector of variables
Z	random vector over z
$\tau(z)$	tree corresponding to setting of z
$Z[t]$	subset of random variables that occur in tree t
$\zeta(t)$	setting of variables in $Z[t]$
$Z[\neg t]$	subset of random variables that do not occur in t

be adapted to other product distributions of $P_z(Z[\neg t] \mid Z[t] = \zeta(t))$.) This choice of $P(Z[\neg t] \mid Z[t] = \zeta(t))$ determines a unique distribution for P_z:

$$\begin{aligned} P_z(Z = z) &= P_t(\tau(z))\, P_z(Z[\neg\tau(z)] = z[\neg\tau(z)] \mid Z[\tau(z)] = z[\tau(z)]) \\ &= P_t(\tau(z)) \prod_{v \in z[\neg\tau(z)]} \frac{1}{\deg(v)} \end{aligned} \tag{2}$$

where $\deg(v)$ is the number of possible values for v.

We proceed by designing a Gibbs sampler for this P_z. The sampler resamples variables from z one at a time, according to the joint probability $P_z(z)$ for each alternative. The set of possible values for z_n at a node n having m incoming hyperedges consists of the hyperedges e_j, $1 \le j \le m$. Let s_j be the vector z with the value of z_n changed to e_j. Note that the $\tau(s_j)$'s only differ at nodes below n.

Let $z[\text{in}(n)]$ be the vector consisting of the variables at nodes below n (that is, contained in subtrees rooted at n, or "inside" n) in the forest, and let $z[\overline{\text{in}}(n)]$ be the vector consisting of variables not under node n. Thus the vectors $z[\text{in}(n)]$ and $z[\overline{\text{in}}(n)]$ partition the complete vector z. We will use the notation $z[t \cap \text{in}(n)]$ to represent the vector of variables from z that are both in a tree t and under a node n.

For Gibbs sampling, we need to compute the relative probabilities of s_j's. We now consider the two terms of Equation (2) in this setting. Because of our requirement that P_z correspond to the desired distribution P_t, the first term of Equation (2) can be computed, up to a normalization constant, by evaluating our model over trees (Equation (1)).

The second term of Equation (2) is a product of uniform distributions that can be decomposed into nodes below n and all other nodes:

$$P_z(Z[\neg t] = z[\neg t] \mid Z[t] = z[t]) = \prod_{v \in z[\neg t \cap \text{in}(n)]} \frac{1}{\deg(v)} \prod_{v \in z[\neg t \cap \overline{\text{in}}(n)]} \frac{1}{\deg(v)} \tag{3}$$

where $t = \tau(z)$. Recall that $\tau(s_j)$'s only differ at vertices below n and hence

$$P_z(Z[\neg t] = z[\neg t] \mid Z[t] = z[t]) \propto \prod_{v \in z[\neg t \cap \text{in}(n)]} \frac{1}{\deg(v)} \tag{4}$$

where we emphasize that $\propto$ refers to the relative probabilities of the s_j's, which correspond to the options from which the Gibbs sampler chooses at a given step. We can manipulate Equation (4) into a more computationally convenient form by multiplying by the $\deg(v)$ term for each node v inside n. Because the terms for all nodes not included in the current tree cancel each other out, we are left with:

$$P_z(Z[\neg t] = z[\neg t] \mid Z[t] = z[t]) \propto \prod_{v \in z[t \cap \text{in}(n)]} \deg(v) \tag{5}$$

Note that we only need to consider the nodes $z[t]$ in the current tree, without needing to examine the remainder of the forest at all.

Substituting Equation (5) into Equation (2) gives a simple update rule for use in our Gibbs sampler:

$$P_z(Z^{(i+1)} = s_j \mid Z^{(i)} = z, n \text{ is updated}) \propto P_t(\tau(s_j)) \prod_{v \in z[\tau(s_j) \cap \text{in}(n)]} \deg(v) \tag{6}$$

To make a step of the Markov chain, we compute the right-hand side of Equation (6) for every s_j and then choose the next state of the chain $Z^{(i+1)}$ from the corresponding distribution on the s_j's. The second term, which we refer to as the **density factor**, is equal to the total number of trees in the forest under node n. This factor compensates for the bias introduced by forests that are bushier in some places than in others, as in the example of Figure 1. A related factor, defined on graphs rather than hypergraphs, can be traced back as far as Knuth (1975), who wished to estimate the sum of values at all nodes in a large tree by sampling a small number of the possible paths from the root to the leaves. Knuth sampled paths uniformly and independently, rather than using a continuously evolving Markov chain as in our Gibbs sampler.

2.2 Sampling Schedule

In a standard Gibbs sampler, updates are made iteratively to each variable $z_1, \ldots, z_N$, and this general strategy can be applied in our case. However, it may be wasteful to continually update variables that are not used by the current tree and are unlikely to be used by any tree. We propose an alternative sampling schedule consisting of sweeping from the root of the current tree down to its leaves, resampling variables at each node in the current tree as we go. If an update changes the structure of the current tree, the sweep continues along the new tree structure. This strategy is shown in Algorithm 1, where $v(z, i)$ denotes the ith variable in a top–down ordering of the variables of the current tree $\tau(z)$. The top–down ordering may be depth-first or breadth-first, among other possibilities, as long as the variables at each node have lower indices than the variables at the node's descendants in the current tree.

To show that this sampling schedule will converge to the desired distribution over trees, we will first show that P_z is a stationary distribution for the transition defined by a single step of the sweep:

Lemma 1
For any setting of variables z, any top–down ordering $v(z, i)$, and any i, updating variable $z_{v(z,i)}$ according to Equation (6) is stationary with respect to the distribution P_z defined by Equation (2).

Algorithm 1 Sampling algorithm

Require: A function $v(z, i)$ returning the index of the ith variable of z in a top-down ordering of the variables of the tree $\tau(z)$.

```
i ← 1
while i ≤ |Z[τ(z)]| do                ▷ Until last node of current tree.
    Resample z_{v(z,i)} according to Equation (6)
    i ← i + 1
end while
```

Proof
We will show that each step of the sweep is stationary for P_z by showing that it satisfies detailed balance. Detailed balance is the condition that, on average, for each pair of states z and z', the number of transitions between the two states is the same in either direction:

$$P_z(Z = z)P(Z^{(i+1)} = z' \mid Z^{(i)} = z) = P_z(Z = z')P(Z^{(i+1)} = z \mid Z^{(i)} = z') \tag{7}$$

where $P(Z^{(i+1)} = z' \mid Z^{(i)} = z)$ is the transition performed by one step of the sweep:

$$P(Z^{(i+1)} = z' \mid Z^{(i)} = z) = \begin{cases} \frac{P_z(Z=z')}{\sum_{z''} P_z(Z=z'')I(z''_{\neg v(z,i)}=z_{\neg v(z,i)})} & \text{if } z'_{\neg v(z,i)} = z_{\neg v(z,i)} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

It is important to observe that, because the resampling step only changes the tree structure below the ith node, the ith node in the new tree remains the same node. That is, after making an update from z to z', $v(z,i) = v(z',i)$, and, mathematically:

$$\begin{aligned} z'_{\neg v(z,i)} = z_{\neg v(z,i)} &\Leftrightarrow v(z,i) = v(z',i) \wedge z'_{\neg v(z',i)} = z_{\neg v(z,i)} \\ &\Leftrightarrow z'_{\neg v(z',i)} = z_{\neg v(z',i)} \end{aligned}$$

Thus, the condition in Equation (8) is symmetric in z and z', and we define the predicate $\text{match}(z, z', i)$ to be equivalent to this condition. Substituting Equation (8) into the left-hand side of Equation (7), we have:

$$P_z(Z = z)P(Z^{(i+1)} = z' \mid Z^{(i)} = z) = \begin{cases} \frac{P_z(Z=z)P_z(Z=z')}{\sum_{z''} P_z(Z=z'')I(z''_{\neg v(z,i)}=z_{\neg v(z,i)})} & \text{if match}(z, z', i) \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

By symmetry of the righthand side of Equation (9) in z and z', we see that Equation (7) is satisfied. Because detailed balance implies stationarity, P_z is a stationary distribution of $P(Z^{(i+1)} = z' \mid Z^{(i)} = z)$. ■

This lemma allows us to prove the correctness of our main algorithm:

Theorem 1
For any top–down sampling schedule $v(z,i)$, and any desired distribution over trees P_t that assigns non-zero probability to all trees in the forest, Algorithm 1 will converge to P_t.

Proof
Because P_z is stationary for each step of the sweep, it is stationary for one entire sweep from top to bottom.

To show that the Markov chain defined by an entire sweep is ergodic, we must show that it is aperiodic and irreducible. It is aperiodic because the chain can stay in the same configuration with non-zero probability by selecting the same setting for each variable in the sweep. The chain is irreducible because any configuration can be reached in a finite number of steps by sorting the variables in topological order bottom–up in the forest, and then, for each variable, executing one sweep that selects a tree that includes the desired variable with the desired setting.

Because P_z is stationary for the chain defined by entire sweeps, and this chain is ergodic, the chain will converge to P_z. Because Equation (2) guarantees that $P_z(\tau(Z) = t) = P_t(t)$, convergence to P_z implies convergence to P_t. ■

2.3 Sampling Composed Rules

Our approach to sampling was motivated by a desire to learn TSG-style grammars, where one grammar rule is the composition of a number of hyperedges in the forest. We extend our sampling algorithm to handle this problem by using the same methods that are used to learn a TSG from a single, fixed tree (Cohn, Goldwater, and Blunsom 2009; Post and Gildea 2009). We attach a binary variable to each node in the forest indicating whether the node is a boundary between two TSG rules, or is internal to a single TSG rule. Thus, the complete set of variables used by the sampler, z, now consists of two variables at each node in the forest: one indicating the incoming hyperedge, and one binary boundary variable. The proof of Section 2.1 carries through, with each new binary variable v having $\deg(v) = 2$ in Equation (2). As before, the current setting of z partitions z into two sets of variables, those used in the current tree, $z[\tau(z)]$, and those outside the current tree, $z[\neg\tau(z)]$. Given a fixed assignment to z, we can read off both the current tree and its segmentation into TSG rules. We modify the tree probability of Equation (1) to be a product over TSG rules r:

$$P_t(t) \propto \prod_{r \in t} w(r) \tag{10}$$

in order to emphasize that grammar rules are no longer strictly equivalent to hyperedges in the forest. We modify the sampling algorithm of Algorithm 1 to make use of this definition of P_t and to resample both variables at the current node. The incoming hyperedge variable is resampled according to Equation (6), while the segmentation variable is simply resampled according to P_t, as the update does not change the sets $z[\tau(z)]$ and $z[\neg\tau(z)]$.

The proof that the sampling algorithm converges to the correct distribution still applies in the TSG setting, as it makes use of the partition of z into $z[\tau(z)]$ and $z[\neg\tau(z)]$, but does not depend on the functional form of the desired distribution over trees P_t.

3. Phrase Decomposition Forest

In the remainder of this article, we will apply the algorithm developed in the previous section to the problem of learning rules for machine translation in the context of a Hiero-style, SCFG-based system. As in Hiero, our grammars will make use of a single nonterminal X, and will contain rules with a mixture of nonterminals and terminals on the right-hand side, with at most two nonterminal occurrences in the right-hand side of a rule. In general, many overlapping rules of varying sizes are consistent with the input word alignments, meaning that we must address a type of segmentation problem in order to learn rules of the right granularity. Given the restriction to two right-hand side nonterminals, the maximum number of rules that can be extracted from an input sentence pair is $O(n^{12})$ in the sentence length, because the left and right boundaries of the left-hand side (l.h.s.) nonterminal and each of the two right-hand side nonterminals can take $O(n)$ positions in each of the two languages. This complexity leads us to explore sampling algorithms, as dynamic programming approaches are likely to be

prohibitively slow. In this section, we show that the problem of learning rules can be analyzed as a problem of identifying tree fragments of unknown size and shape in a forest derived from the input word alignments for each sentence. These tree fragments are similar to the tree fragments used in TSG learning. As in TSG learning, each rule of the final grammar consists of some number of adjacent, minimal tree fragments: one-level treebank expansions in the case of TSG learning and minimal SCFG rules, defined subsequently, in the case of translation. The internal structure of TSG rules is used during parsing to determine the final tree structure to output, and the internal structure of machine translation rules will not be used at decoding time. This distinction is irrelevant during learning. A more significant difference from TSG learning is that the sets of minimal tree fragments in our SCFG application come not from a single, known tree, but rather from a forest representing the set of bracketings consistent with the input word alignments.

We now proceed to precisely define this **phrase decomposition forest** and discuss some of its theoretical properties. The phrase decomposition forest is designed to extend the phrase decomposition tree defined by Zhang, Gildea, and Chiang (2008) in order to explicitly represent each possible minimal rule with a hyperedge.

A **span** $[i, j]$ is a set of contiguous word indices $\{i, i+1, \ldots, j-1\}$. Given an aligned Chinese–English sentence pair, a **phrase** n is a pair of spans $n = ([i_1, j_1], [i_2, j_2])$ such that Chinese words in positions $[i_1, j_1]$ are aligned only to English words in positions $[i_2, j_2]$, and vice versa. A **phrase forest** $H = \langle V, E \rangle$ is a hypergraph made of a set of hypernodes V and a set of hyperedges E. Each node $n = ([i_1, j_1], [i_2, j_2]) \in V$ is a **tight** phrase as defined by Koehn, Och, and Marcu (2003), namely, a phrase containing no unaligned words at its boundaries. A phrase $n = ([i_1, j_1], [i_2, j_2])$ **covers** $n' = ([i'_1, j'_1], [i'_2, j'_2])$ if

$$i_1 \leq i'_1 \wedge j'_1 \leq j_1 \wedge i_2 \leq i'_2 \wedge j'_2 \leq j_2$$

Note that every phrase covers itself. It follows from the definition of phrases that if $i_1 \leq i'_1 \wedge j'_1 \leq j_1$, then $i_2 \leq i'_2 \wedge j'_2 \leq j_2$. That means we can determine phrase coverage by only looking at one language side of the phrases. We are going to use this property to simplify the discussion of our proofs. We also define coverage between two sets of phrases. Given two sets of phrases T and T', we say T' covers T if for all $t \in T$, there exists a $t' \in T'$ such that t' covers t. We say that two phrases **overlap** if they intersect, but neither covers the other.

If two phrases $n = ([i_1, j_1], [i_2, j_2])$ and $n' = ([i'_1, j'_1], [i'_2, j'_2])$ intersect, we can take the union of the two phrases by taking the union of the source and target language spans, respectively. That is, $n_1 \cup n_2 = ([i_1, j_1] \cup [i'_1, j'_1], [i_2, j_2] \cup [i'_2, j'_2])$. An important property of phrases is that if two phrases intersect, their union is also a phrase. For example, given that *have a date with her* and *with her today* are both valid phrases in Figure 2, *have a date with her today* must also be a valid phrase. Given a set T of phrases, we define the **union closure** of the phrase set T, denoted $\bigcup^*(T)$, to be constructed by repeatedly joining intersecting phrases until there are no intersecting phrases left.

Each edge in E, written as $T \rightarrow n$, is made of a set of non-intersecting tail nodes $T \subset V$, and a single head node $n \in V$ that covers each tail node. Each edge is an SCFG rule consistent with the word alignments. Each tail node corresponds to a right-hand-side nonterminal in the SCFG rule, and any position included in n but not included in any tail node corresponds to a right-hand-side terminal in the SCFG rule. For example, given the aligned sentence pair of Figure 2, the edge $\{([3,4],[5,6]), ([5,6],[3,4])\} \rightarrow ([2,6],[1,6])$, corresponds to a SCFG rule $X \rightarrow$ 和 X_1 有 X_2, have a X_2 with X_1.

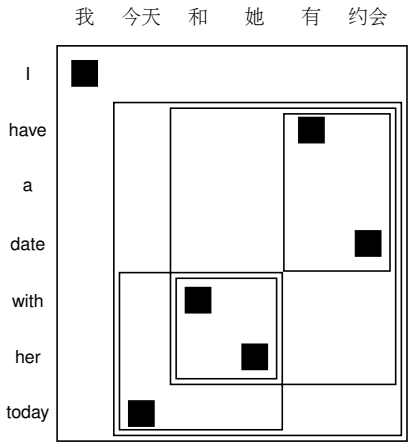

Figure 2
Example word alignment, with boxes showing valid phrase pairs. In this example, all individual alignment points are also valid phrase pairs.

For the rest of this section, we assume that there are no unaligned words. Unaligned words can be temporarily removed from the alignment matrix before building the phrase decomposition forest. After extracting the forest, they are put back into the alignment matrix. For each derivation in the phrase decomposition forest, an unaligned word appears in the SCFG rule whose left-hand side corresponds to the lowest forest node that covers the unaligned word.

Definition 1
An edge $T \rightarrow n$ is **minimal** if there does not exist another edge $T' \rightarrow n$ such that T' covers T.

A minimal edge is an SCFG rule that cannot be decomposed by factoring out some part of its right-hand side as a separate rule. We define a phrase decomposition forest to be made of all phrases from a sentence pair, connected by all minimal SCFG rules. A phrase decomposition forest compactly represents all possible SCFG rules that are consistent with word alignments. For the example word alignment shown in Figure 2, the phrase decomposition forest is shown in Figure 3. Each boxed phrase in Figure 2 corresponds to a node in the forest of Figure 3, and hyperedges in Figure 3 represent ways of building phrases out of shorter phrases.

A phrase decomposition forest has the important property that any SCFG rule consistent with the word alignment corresponds to a contiguous fragment of some complete tree found in the forest. For example, the highlighted tree fragment of the forest in Figure 3 corresponds to the SCFG rule:

$$X \rightarrow \text{和 } X_2 \text{ 有 } X_1,\ \text{have a } X_1 \text{ with } X_2$$

Thus any valid SCFG rule can be formed by selecting a set of adjacent hyperedges from the forest and composing the minimal SCFG rules specified by each hyperedge. We will apply the sampling algorithm developed in Section 2 to this problem of selecting hyperedges from the forest.

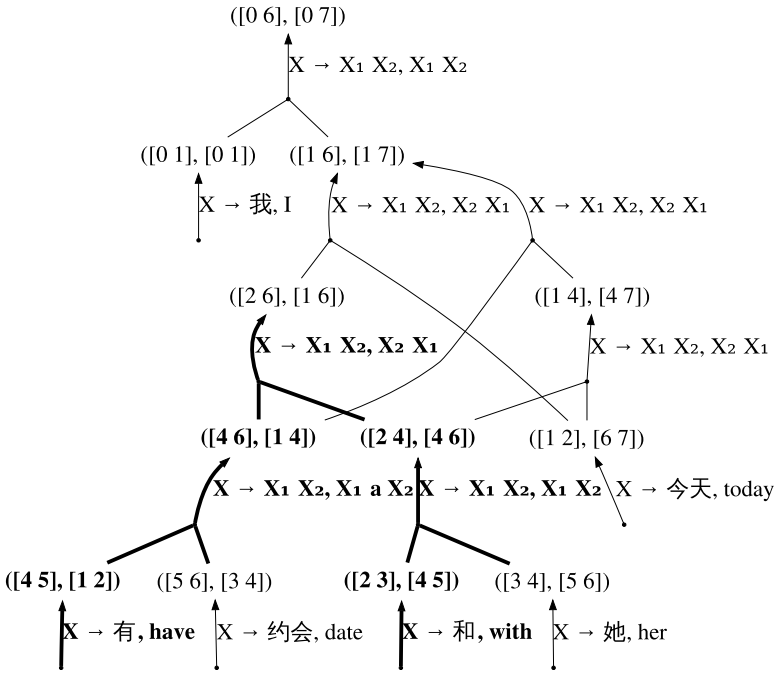

Figure 3
A phrase decomposition forest extracted from the sentence pair ⟨我 今天 和 她 有 约会, *I have a date with her today*⟩. Each edge is a minimal SCFG rule, and the rules at the bottom level are phrase pairs. Unaligned word "a" shows up in the rule $X \rightarrow X_1X_2, X_1aX_2$ after unaligned words are put back into the alignment matrix. The highlighted portion of the forest shows an SCFG rule built by composing minimal rules.

The structure and size of phrase decomposition forests are constrained by the following lemma:

Lemma 2
When there exists more than one minimal edge leading to the same head node $n = ([i_1, j_1], [i_2, j_2])$, each of these minimal edges is a binary split of phrase pair n, which gives us either a straight or inverted binary SCFG rule with no terminals.

Proof
Suppose that there exist two minimal edges $T_1 \rightarrow n$ and $T_2 \rightarrow n$ leading to node n. Consider the node set we get by taking the union closure of the tail nodes in T_1 and T_2:

$$\bigcup^*(T_1 \cup T_2) \rightarrow n$$

Figure 4 shows two cases of this construction. We show only the spans on the source language side, which is enough to determine coverage properties. Let n have span $[i, j]$ on the source side. In the first case (left), $\bigcup^*(T_1 \cup T_2) \neq \{n\}$. We know $\bigcup^*(T_1 \cup T_2) \rightarrow n$ is also a valid edge because the unions of intersecting phrases are phrases, too. By the definition of union closure, $\bigcup^*(T_1 \cup T_2)$ covers both T_1 and T_2. Therefore $T_1 \rightarrow n$ and

$T_2 \rightarrow n$ cannot both be minimal. In the second case (right), $\bigcup^*(T_1 \cup T_2) = \{n\}$. This means that the phrases in $T_1 \cup T_2$ overlap one another in a chain covering the entire span of n. There must exist a phrase $n_1 = [i, k_1]$ in T_1 or T_2 that begins at the left boundary i of n. Without loss of generality, assume that $n_1 \in T_1$. There must exist another phrase $n_2 = [k_2, j_2] \in T_2$ that overlaps with n such that $k_2 < k_1$ and $j_2 > k_1$. The span $[k_2, j]$ is a valid phrase, because it consists of the union closure of all phrases that begin to the right of k_2:

$$\bigcup^* \{[i', j'] \mid [i', j'] \in T_1 \cup T_2 \wedge i' \geq k_2\} = \{[k_2, j]\}$$

We also know that $n_1 - n_2 = [i, k_2]$ is a valid phrase because the difference of two overlapping phrases is also a valid phrase. Therefore k_2 is a valid binary split point of n, which means that either T_2 is an edge formed by this binary split, or T_2 is not minimal. The span $[k_2, j] - n_1 = [k_1, j]$ is also a valid phrase formed by taking the difference of two overlapping phrases, which makes k_1 a valid binary split point for n. This makes T_1 either an edge formed by the binary split at k_1, or not a minimal phrase. Thus, whenever we have two minimal edges, both consist of a binary split. ■

Another interesting property of phrase decomposition forests relates to the length of **derivations**. A derivation is a tree of minimal edges reaching from a given node all the way down to the forest's terminal nodes. The **length** of a derivation is the number of minimal edges it contains.

Lemma 3
All derivations under a node in a phrase decomposition forest have the same length.

Proof
This is proved by induction. As the base case, all the nodes at the bottom of the phrase decomposition forest have only one derivation of length 1. For the induction step, we consider the two possible cases in Lemma 2. The case where a node n has only a single edge underneath is trivial. It can have only one derivation length because the children under that single edge already do. For the case where there are multiple valid binary splits for a node n at span (i, j), we assume the split points are $k_1, \ldots, k_s$. Because the intersection of two phrases is also a phrase, we know that spans $(i, k_1), (k_1, k_2), \ldots, (k_s, j)$ are all valid phrases, and so is any concatenation of consecutive phrases in these spans. Any derivation in this sub-forest structure leading from these $s + 1$ spans to n has length s, which completes the proof under the assumption of the induction. ■

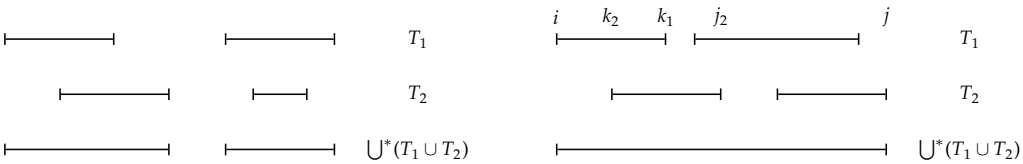

Figure 4
Sketch of proof for Lemma 2. In the first case, $\bigcup^*(T_1 \cup T_2)$ consists of more than one span, or consists of one span that is strictly smaller than n. In the second case, $\bigcup^*(T_1 \cup T_2) = \{n\}$.

Because all the different derivations under the same node in a minimal phrase forest contain the same number of minimal rules, we call that number the **level** of a node. The fact that nodes can be grouped by levels forms the basis of our fast iterative sampling algorithm as described in Section 5.3.

3.1 Constructing the Phrase Decomposition Forest

Given a word-aligned sentence pair, a phrase decomposition tree can be extracted with a shift-reduce algorithm (Zhang, Gildea, and Chiang 2008). Whereas the algorithm of Zhang, Gildea, and Chiang (2008) constructs a single tree which compactly represents the set of possible phrase trees, we wish to represent the set of all trees as a forest. We now describe a bottom–up parsing algorithm, shown in Algorithm 2, for building this forest. The algorithm considers all spans (i, j) in order of increasing length. The CYK-like loop over split points k (line 10) is only used for the case where a phrase can be decomposed into two phrases, corresponding to a binary SCFG rule with no right-hand side terminals. By Lemma 2, this is the only source of ambiguity in constructing

Algorithm 2 The CYK-like algorithm for building a phrase decomposition forest from word-aligned sentence pair $\langle f, e \rangle$.

```
1: Extract all phrase pairs in the form of ([i1, j1], [i2, j2])
2: Build a forest node for each phrase pair, and let n(i, j) be the node corresponding to the phrase
   pair whose source side is [i, j]
3: for s = 1 ... |f| do
4:     for i = 0 ... |f| − s do
5:         j ← i + s
6:         if n(i, j) exists then
7:             continue
8:         end if
9:         split ← 0
10:        for k = i + 1 ... j − 1 do
11:            if both n(i, k) and n(k, j) exist then
12:                add edge {n(i, k), n(k, j)} → n(i, j)
13:                split ← split + 1
14:            end if
15:        end for
16:        if split = 0 then
17:            T ← ∅
18:            l ← i
19:            while l < j do
20:                l′ ← l + 1
21:                for m ← j ... l do
22:                    if n(l, m) exists then
23:                        T ← T ∪ n(l, m)
24:                        l′ ← m
25:                        break
26:                    end if
27:                end for
28:                l ← l′
29:            end while
30:            add edge T → n(i, j)
31:        end if
32:    end for
33: end for
```

phrase decompositions. When no binary split is found (line 16), a single hyperedge is made that connects the current span with all its maximal children. (A child is maximal if it is not itself covered by another child.) This section can produce SCFG rules with more than two right-hand side nonterminals, and it also produces any rules containing both terminals and nonterminals in the right-hand side. Right-hand side nonterminals correspond to previously constructed nodes $n(l, m)$ in line 23, and right-hand side terminals correspond to advancing a position in the string in line 20.

The running time of the algorithm is $O(n^3)$ in terms of the length of the Chinese sentence f. The size of the resulting forests depends on the input alignments. The worst case in terms of forest size is when the input consists of a monotonic, one-to-one word alignment. In this situation, all (i, k, j) tuples correspond to valid hyperedges, and the size of the output forest is $O(n^3)$. At the other extreme, when given a non-decomposable permutation as an input alignment, the output forest consists of a single hyperedge. In practice, given Chinese–English word alignments from GIZA++, we find that the resulting forests are highly constrained, and the algorithm's running time is negligible in our overall system. In fact, we find it better to rebuild every forest from a word alignment every time we re-sample a sentence, rather than storing the hypergraphs across sampling iterations.

4. Comparison of Sampling Methods

To empirically verify the sampling methods presented in Section 2, we construct phrase decomposition forests over which we try to learn composed translation rules. In this section, we use a simple probability model for the tree probability P_t in order to study the convergence behavior of our sampling algorithm. We will use a more sophisticated probability model for our end-to-end machine translation experiments in Section 5. For studying convergence, we desire a simpler model with a probability that can be evaluated in closed form.

4.1 Model

We use a very basic generative model based on a Dirichlet process defined over composed rules. The model is essentially the same as the TSG model used by Cohn, Goldwater, and Blunsom (2009) and Post and Gildea (2009).

We define a single Dirichlet process over the entire set of rules. We draw the rule distribution G from a Dirichlet process, and then rules from G.

$$G \mid \alpha, P_0 \sim \text{Dir}(\alpha, P_0)$$
$$r \mid G \sim G$$

For the base distribution P_0, we use a very simple uniform distribution where all rules of the same size have equal probability:

$$P_0(r) = {V_f}^{-|r_f|} {V_e}^{-|r_e|}$$

where V_f is the vocabulary size of source language, and $|r_f|$ is the length of the source side of the rule r. Integrating over G, we get a Chinese restaurant process for the Dirichlet process. Customers in the Chinese restaurant analogy represent translation rule instances in the machine translation setting, and tables represent rule types. The

Chinese restaurant has an infinite number of tables, and customers enter one by one and choose a table to sit at. Let z_i be the table chosen by ith customer. Then, the probability of the customer choosing a table which is already occupied by customers who entered the restaurant previously, or a new table, is given by following equations:

$$P(z_i = t \mid \mathbf{z}_{-\mathbf{i}}) = \begin{cases} \frac{n_t}{i-1+\alpha} & 1 \leq t \leq T \\ \frac{\alpha}{i-1+\alpha} & t = T+1 \end{cases}$$

where $\mathbf{z}_{-\mathbf{i}}$ is the current seating arrangement, t is the index of the table, n_t is the number of customers at the table t, and T is the total number of occupied tables in the restaurant. In our model, a table t has a label indicating to which rule r the table is assigned. The label of a table is drawn from the base distribution P_0.

If we marginalize over tables labeled with the same rule, we get the following probability of choosing r given the current analysis $\mathbf{z}_{-\mathbf{i}}$ of the data:

$$P(r_i = r \mid \mathbf{z}_{-\mathbf{i}}) = \frac{n_r + \alpha P_0(r)}{n + \alpha} \tag{11}$$

where n_r is the number of times rule r has been observed in $\mathbf{z}_{-\mathbf{i}}$, and n is total number of rules observed in $\mathbf{z}_{-\mathbf{i}}$.

4.2 Sampling Methods

We wish to sample from the set of possible decompositions into rules, including composed rules, for each sentence in our training data. We follow the top–down sampling schedule discussed in Section 2 and also implement tree-level rejection sampling as a baseline.

Our rejection sampling baseline is a form of Metropolis-Hastings where a new tree t is resampled from a simple proposal distribution $Q(t)$, and then either accepted or rejected according the Metropolis-Hastings acceptance rule, as shown in Algorithm 3. As in Algorithm 1, we use $v(z, i)$ to denote a top–down ordering of forest variables. As in all our experiments, P_t is the current tree probability conditioned on the current trees for all other sentences in our corpus, using Equation (11) as the rule probability in Equation (10).

Our proposal distribution samples each variable with uniform probability working top–down through the forest. The proposal distribution for an entire tree is thus:

$$Q(t) = \prod_{w \in z[t]} \frac{1}{\deg(w)}$$

This does not correspond to a uniform distribution over entire trees for the reasons discussed in Section 2. However, the Metropolis-Hastings acceptance probability corrects for this, and thus the algorithm is guaranteed to converge to the correct distribution in the long term. We will show that, because the proposal distribution does not re-use any of the variable settings from the current tree, the rejection sampling algorithm converges more slowly in practice than the more sophisticated alternative described in Section 2.2.

We now describe in more detail our implementation of the approach of Section 2.2. We define two operations on a hypergraph node n, SAMPLECUT and SAMPLEEDGE, to change the sampled tree from the hypergraph. SAMPLECUT(n) chooses whether n is a

Algorithm 3 Metropolis-Hastings sampling algorithm.

Require: A function $v(z, i)$ returning the index of the ith variable of z in a top-down ordering of the variables of the tree $\tau(z)$.

$i \leftarrow 1$
while $i < |Z[\tau(z)]|$ **do**
  Sample $z^{new}_{v(z,i)}$ according to uniform$(z^{new}_{v(z,i)})$
  $i \leftarrow i + 1$
end while
$z \leftarrow \begin{cases} z^{new} & \text{w/prob min} \left\{1, \frac{P_t(t(z^{new}))Q(t(z^{old}))}{P_t(t(z^{old}))Q(t(z^{new}))}\right\} \\ z^{old} & \text{otherwise} \end{cases}$

segmentation point or not, deciding if two rules should merge, while SAMPLEEDGE(n) chooses a hyperedge under n, making an entire new subtree. Algorithm 4 shows our implementation of Algorithm 1 in terms of tree operations and the sampling operations SAMPLEEDGE(n) and SAMPLECUT(n).

4.3 Experiments

We used a Chinese–English parallel corpus available from the Linguistic Data Consortium (LDC), composed of newswire text. The corpus consists of 41K sentence pairs, which is 1M words on the English side. We constructed phrase decomposition forests with this corpus and ran the top–down sampling algorithm and the rejection sampling algorithm described in Section 4.2 for one hundred iterations. We used $\alpha = 100$ for every experiment. The likelihood of the current state was calculated for every iteration. Each setting was repeated five times, and then we computed the average likelihood for each iteration.

Figure 5 shows a comparison of the likelihoods found by rejection sampling and top–down sampling. As expected, we found that the likelihood converged much more quickly with top–down sampling. Figure 6 shows a comparison between two different versions of top–down sampling: the first experiment was run with the density factor described in Section 2, Equation (6), and the second one was run without the density factor. The density factor has a much smaller effect on the convergence of our algorithm than does the move from rejection sampling to top–down sampling, such that the difference between the two curves shown in Figure 6 is not visible at the scale of Figure 5. (The first ten iterations are omitted in Figure 6 in order to highlight the difference.) The small difference is likely due to the fact that our trees are relatively evenly balanced,

Algorithm 4 Top–down sampling algorithm.

queue.push(root)
while queue is not empty **do**
  n = queue.pop()
  SAMPLEEDGE(n)
  SAMPLECUT(n)
  for each child c of node n **do**
    queue.push(c)
  end for
end while

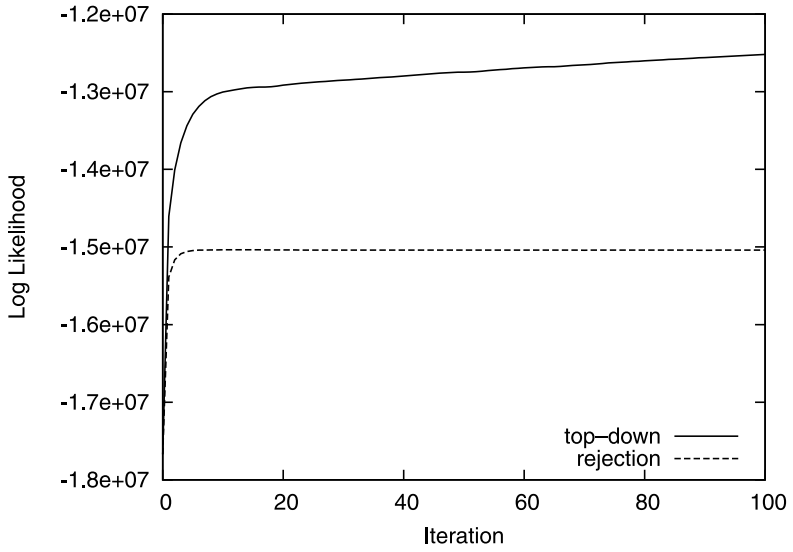

Figure 5
Likelihood graphs for rejection sampling and top–down sampling.

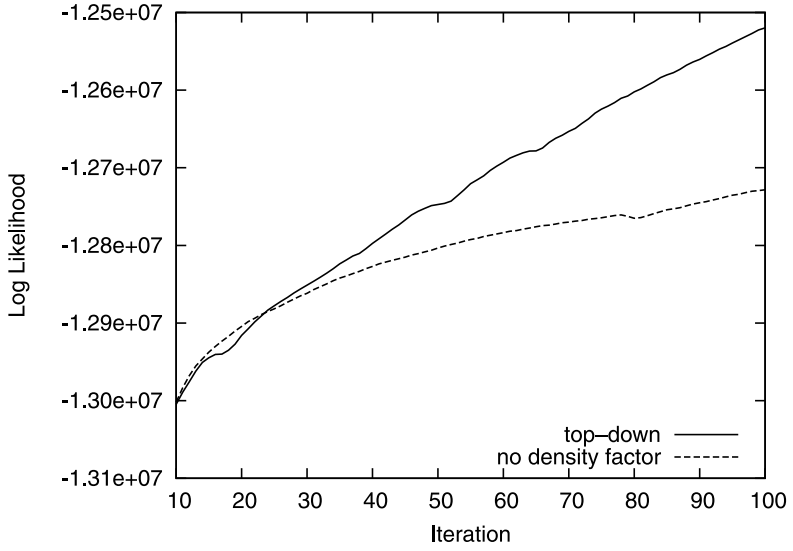

Figure 6
Likelihood graphs for top–down sampling with and without density factor. The first ten iterations are omitted to highlight the difference.

such that the ratio of the density factor for two trees is not significant in comparison to the ratio of their model probabilities. Nevertheless, we do find higher likelihood states with the density factor than without it. This shows that, in addition to providing a theoretical guarantee that our Markov chain converges to the desired distribution P_t in the limit, the density factor also helps us find higher probability trees in practice.

5. Application to Machine Translation

The results of the previous section demonstrate the performance of our algorithm in terms of the probabilities of the model it is given, but do not constitute an end-to-end application. In this section we demonstrate its use in a complete machine translation system, using the SCFG rules found by the sampler in a Hiero-style MT decoder. We discuss our approach and how it relates to previous work in machine translation in Section 5.1 before specifying the precise probability model used for our experiments in Section 5.2, discussing a technique to speed-up the model's burn-in in Section 5.3, and describing our experiments in Section 5.4.

5.1 Approach

A typical pipeline for training current statistical machine translation systems consists of the following three steps: word alignment, rule extraction, and tuning of feature weights. Word alignment is most often performed using the models of Brown et al. (1993) and Vogel, Ney, and Tillmann (1996). Phrase extraction is performed differently for phrase-based (Koehn, Och, and Marcu 2003), hierarchical (Chiang 2005), and syntax-based (Galley et al. 2004) translation models, whereas tuning algorithms are generally independent of the translation model (Och 2003; Chiang, Marton, and Resnik 2008; Hopkins and May 2011).

Recently, a number of efforts have been made to combine the word alignment and rule extraction steps into a joint model, with the hope both of avoiding some of the errors of the word-level alignment, and of automatically learning the decomposition of sentence pairs into rules (DeNero, Bouchard-Cote, and Klein 2008; Blunsom et al. 2009; Blunsom and Cohn 2010a; Neubig et al. 2011). This approach treats both word alignment and rule decomposition as hidden variables in an EM-style algorithm. While these efforts have been able to match the performance of systems based on two successive steps for word alignment and rule extraction, they have generally not improved performance enough to become widely adopted. One possible reason for this is the added complexity and in particular the increased computation time when compared to the standard pipeline. The accuracy of word-level alignments from the standard GIZA++ package has proved hard to beat, in particular when large amounts of training data are available.

Given this state of affairs, the question arises whether static word alignments can be used to guide rule learning in a model which treats the decomposition of a sentence pair into rules as a hidden variable. Such an approach would favor rules which are consistent with the other sentences in the data, and would contrast with the standard practice in Hiero-style systems of simply extracting all overlapping rules consistent with static word alignments. Constraining the search over rule decomposition with word alignments has the potential to significantly speed up training of rule decomposition models, overcoming one of the barriers to their widespread use. Rule decomposition models also have the benefit of producing much smaller grammars than are achieved when extracting all possible rules. This is desirable given that the size of translation grammars is one of the limiting computational factors in current systems, necessitating elaborate strategies for rule filtering and indexing.

In this section, we apply our sampling algorithm to learn rules for the Hiero translation model of Chiang (2005). Hiero is based on SCFG, with a number of constraints on the form that rules can take. The grammar has a single nonterminal, and each rule has

at most two right-hand side nonterminals. Most significantly, Hiero allows rules with mixed terminals and nonterminals on the right-hand side. This has the great benefit of allowing terminals to control re-ordering between languages, but also leads to very large numbers of valid rules during the rule extraction process. We wish to see whether, by adding a learned model of sentence decomposition to Hiero's original method of leveraging fixed word-level alignments, we can learn a small set of rules in a system that is both efficient to train and efficient to decode. Our approach of beginning with fixed word alignments is similar to that of Sankaran, Haffari, and Sarkar (2011), although their sampling algorithm reanalyzes individual phrases extracted with Hiero heuristics rather than entire sentences, and produces rules with no more than one nonterminal on the right-hand side.

Most previous works on joint word alignment and rule extraction models were evaluated indirectly by resorting to heuristic methods to extract rules from learned word alignment or bracketing structures (DeNero, Bouchard-Cote, and Klein 2008; Zhang et al. 2008; Blunsom et al. 2009; Levenberg, Dyer, and Blunsom 2012), and do not directly learn the SCFG rules that are used during decoding. In this article, we work with lexicalized translation rules with a mix of terminals and nonterminals, and we use the rules found by our sampler directly for decoding. Because word alignments are fixed in our model, any improvements we observe in translation quality indicate that our model learns how SCFG rules interplay with each other, rather than fixing word alignment errors.

The problem of rule decomposition is not only relevant to the Hiero model. Translation models that make use of monolingual parsing, such as string-to-tree (Galley et al. 2004), and tree-to-string (Liu, Liu, and Lin 2006), are all known to benefit greatly from learning composed rules (Galley et al. 2006). In the particular case of Hiero rule extraction, although there is no explicit rule composition step, the extracted rules are in fact "composed rules" in the sense of string-to-tree or tree-to-string rule extraction, because they can be further decomposed into smaller SCFG rules that are also consistent with word alignments. Although our experiments only include the Hiero model, the method presented in this article is also applicable to string-to-tree and tree-to-string models, because the phrase decomposition forest presented in Section 3 can be extended to rule learning and extraction of other syntax-based MT models.

5.2 Model

In this section, we describe a generative model based on the Pitman-Yor process (Pitman and Yor 1997; Teh 2006) over derivation trees consisting of composed rules. Bayesian methods have been applied to a number of segmentation tasks in natural language processing, including word segmentation, TSG learning, and learning machine translation rules, as a way of controlling the overfitting produced when Expectation Maximization would tend to prefer longer segments. However, it is important to note that the Bayesian priors in most cases control the size and number of the clusters, but do not explicitly control the size of rules. In many cases, this type of Bayesian prior alone is not strong enough to overcome the preference for longer, less generalizable rules. For example, some previous work in word segmentation (Liang and Klein 2009; Naradowsky and Toutanova 2011) adopts a "length penalty" to remedy this situation. Because we have the prior knowledge that longer rules are less likely to generalize and are therefore less likely to be a good rule, we adopt a similar scheme to control the length of rules in our model.

In order to explicitly control the length of our rules, we generate a rule r in two stages. First, we draw the length of a rule $|r| = \ell$ from a probability distribution defined over positive integers. We use a Poisson distribution:

$$P(\ell; \lambda) = \frac{\lambda^{\ell} e^{-\lambda}}{\ell!}$$

Because of the factorial in the denominator, the Poisson distribution decays quickly as ℓ becomes larger, which is ideal for selecting rule length because we want to encourage learning of shorter rules and learn longer rules only when there is strong evidence for them in the data.

A separate Pitman-Yor process is defined for the rules of each length ℓ. We draw the rule distribution G from a Pitman-Yor process, and then rules of length ℓ are drawn from G.

$$G \mid \alpha, d, P_0 \sim \mathrm{PY}(\alpha, d, P_0)$$
$$r \mid G \sim G$$

The first two parameters, a concentration parameter α and a discount parameter d, control the shape of distribution G by controlling the size and the number of clusters. The label of the cluster is decided by the base distribution P_0. Because our alignment is fixed, we do not need a complex base distribution that differentiates better aligned phrases from others. We use a uniform distribution where each rule of the same size has equal probability. Since the number of possible shorter rules is smaller than that of longer rules, we need to reflect this fact and need to have larger uniform probability for shorter rules and smaller uniform probability for longer rules. We reuse the Poisson probability for the base distribution, essentially assuming that the number of possible rules of length ℓ is $1/P(\ell; \lambda)$.

The Pitman-Yor process gives us the following probability of choosing r of size ℓ given the current analysis $\mathbf{z}_{-\mathbf{i}}$ of the data:

$$P(r_i = r \mid \ell, \mathbf{z}_{-\mathbf{i}}) = \frac{n_r - T_r d + (T_\ell d + \alpha) P_0(r)}{n_\ell + \alpha}$$

where n_r is the number of times rule r has been observed in $\mathbf{z}_{-\mathbf{i}}$, T_r is the number of tables (in the Chinese restaurant metaphor) labeled r, and n_ℓ is the total number of rules of length ℓ observed in $\mathbf{z}_{-\mathbf{i}}$. Because we have drawn the length of the rule from a Poisson distribution, the rule length probability is multiplied by this equation in order to obtain the probability of the rule under our model.

Keeping track of table assignments during inference requires a lot of book-keeping. In order to simplify the implementation, instead of explicitly keeping track of the number of tables for each rule, we estimate the number of tables using the following equations (Huang and Renals 2010):

$$T_r = n_r^d$$
$$T_\ell = \sum_{r:|r|=\ell} n_r^d$$

In order to encourage learning rules with smaller parsing complexity and rules with mixed terminals and nonterminals, which are useful for replicating re-orderings that are seen in the data, we made use of the concept of **scope** (Hopkins and Langmead 2010) in our definition of rule length. The scope of a rule is defined as the number of pairs of adjacent nonterminals in the source language right-hand side plus the number of nonterminals at the beginning or end of the source language right-hand side. For example,

$$X \rightarrow f_1 X_1 X_2 f_2 X_3,\ X_1 e_1 X_2 X_3 e_2$$

has scope 2 because X_1 and X_2 are adjacent in the source language and X_3 is at the end of the source language right-hand side. The target side of the rule is irrelevant. The intuition behind this definition is that it measures the number of free indices into the source language string required during parsing, under the assumption that the terminals provide fixed anchor points into the string. Thus a rule of scope of k can be parsed in $O(n^k)$. We define the length of a rule to be the number of terminals in the source and the target side plus the scope of the rule. This is equivalent to counting the total number of symbols in the rule, but only counting a nonterminal if it contributes to parsing complexity. For example, the length of a rule that consists only of two consecutive nonterminals would be 3, and the length of a rule that has two consecutive nonterminals bounded by terminals on both sides would be 3 as well. This definition of rule length encourages rules with mixed terminals and nonterminals over rules with only nonterminals, which tend not to provide useful guidance to the translation process during decoding.

5.3 Stratified Sampling

We follow the same Gibbs sampler introduced in Section 4.2. The SAMPLEEDGE operation in our Gibbs sampler can be a relatively expensive operation, because the entire subtree under a node is being changed during sampling. We observe that in a phrase decomposition forest, lexicalized rules, which are crucial to translation quality, appear at the bottom level of the forest. This lexicalized information propagates up the forest as rules get composed. It is reasonable to constrain initial sampling iterations to work only on those bottom level nodes, and then gradually lift the constraint. This not only makes the sampler much more efficient, but also gives it a chance to focus on getting better estimates of the more important parameters, before starting to consider nodes at higher levels, which correspond to rules of larger size. Fortunately, as mentioned in Section 3, each node in a phrase decomposition forest already has a unique level, with level 1 nodes corresponding to minimal phrase pairs. We design the sampler to use a stratified sampling process (i.e., sampling level one nodes for K iterations, then level 1 and 2 nodes for K iterations, and so on). We emphasize that when we sample for level 2 nodes, level 1 nodes are also sampled, which means parameters for the smaller rules are given more chance to mix, and thereby settle into a more stable distribution.

In our experiments, running the first 100 iterations of sampling with regular sampling techniques took us about 18 hours. However, with stratified sampling, it took only about 6 hours. We also compared translation quality as measured by decoding with rules from the 100th sample, and by averaging over every 10th sample. Both sampling methods gave us roughly the same translation quality as measured in BLEU. We therefore used stratified sampling throughout our experiments.

5.4 Experiments

We used a Chinese–English parallel corpus available from LDC,[1] composed of newswire text. The corpus consists of 41K sentence pairs, which is 1M words on the English side. We used a 392-sentence development set with four references for parameter tuning, and a 428-sentence test set with four references for testing.[2] The development set and the test set have sentences with less than 30 words. A trigram language model was used for all experiments. BLEU (Papineni et al. 2002) was calculated for evaluation.

5.4.1 Baseline. For our baseline system, we extract Hiero translation rules using the heuristic method (Chiang 2007), with the standard Hiero rule extraction constraints. We use our in-house SCFG decoder for translation with both the Hiero baseline and our sampled grammars. Our features for all experiments include differently normalized rule counts and lexical weightings (Koehn, Och, and Marcu 2003) of each rule. Weights are tuned using Pairwise Ranking Optimization (Hopkins and May 2011) using the baseline grammar and development set, then used throughout the experiments.

Because our sampling procedure results in a smaller rule table, we also establish a no-singleton baseline to compare our results to a simple heuristic method of reducing rule table size. The no-singleton baseline discards rules that occur only once and that have more than one word on the Chinese side during the Hiero rule extraction process, before counting the rules and computing feature scores.

5.4.2 Experimental Settings.

Model parameters. For all experiments, we used $d = 0.5$ for the Pitman-Yor discount parameter, except where we compared the Pitman-Yor process with Dirichlet process ($d = 0$). Although we have a separate Pitman-Yor process for each rule length, we used the same $\alpha = 5$ for all rule sizes in all experiments, including Dirichlet process experiments. For rule length probability, a Poisson distribution where $\lambda = 2$ was used for all experiments.

Sampling. The samples are initialized such that all nodes in a forest are set to be segmented, and a random edge is chosen under each node. For all experiments, we ran the sampler for 100 iterations and took the sample from the last iteration to compare with the baseline. For stratified sampling, we increased the level we sample at every 10th iteration. We also tried "averaging" samples, where samples from every 10th iteration are merged to a single grammar. For averaging samples, we took the samples from the 0th iteration (initialization) to the 70th iteration at every 10th iteration.[3] We decided on the 70th iteration (last iteration of level 7 sampling) as the last iteration because we constrained the sampler not to sample nodes whose span covers more than seven words (for SAMPLECUT only, SAMPLECUT always segments for these nodes), and the likelihood becomes very stable at that point.

1 We randomly sampled our data from various different sources (LDC2006E86, LDC2006E93, LDC2002E18, LDC2002L27, LDC2003E07, LDC2003E14, LDC2004T08, LDC2005T06, LDC2005T10, LDC2005T34, LDC2006E26, LDC2005E83, LDC2006E34, LDC2006E85, LDC2006E92, LDC2006E24, LDC2006E92, LDC2006E24). The language model is trained on the English side of entire data (1.65M sentences, which is 39.3M words).

2 They are from newswire portion of NIST MT evaluation data from 2004, 2005, and 2006.

3 Not including initialization has negligible effect on translation quality.

Rule extraction. Because every tree fragment in the sampled derivation represents a translation rule, we do not need to explicitly extract the rules; we merely need to collect them and count them. However, derivations include purely nonlexical rules that do not conform to the rule constraints of Hiero, and which are not useful for translation. To get rid of this type of rule, we prune every rule that has a scope greater than 2. Whereas Hiero does not allow two adjacent nonterminals in the source side, our pruning criterion allows some rules of scope 2 that are not allowed by Hiero. For example, the following rule (only source side shown) has scope 2 but is not allowed by Hiero:

$$X \rightarrow w_1 X_1 X_2 w_2 X_3$$

In order to see if these rules have any positive or negative effects on translation, we compare a rule set that strictly conforms to the Hiero constraints and a rule set that includes all the rules of scope 2 or less.

5.4.3 Results. Table 2 summarizes our results. As a general test of our probability model, we compare the result from initialization and the 100th sample. The translation performance of the grammar from the 100th iteration of sampling is much higher than that of the initialization state. This shows that states with higher probability in our Markov chain generally do result in better translation, and that the sampling process is able to learn valuable composed rules.

In order to determine whether the composed rules learned by our algorithm are particularly valuable, we compare them to the standard baseline of extracting all rules. The size of the grammar taken from the single sample (100th sample) is only about 9% of the baseline but still produces translation results that are not far worse than the baseline. A simple way to reduce the number of rules in the baseline grammar is to remove all rules that occur only once in the training data and that contain more than a single word on the Chinese side. This "no-singleton" baseline still leaves us with more rules than our algorithm, with translation results between those of the standard baseline and our algorithm.

We also wish to investigate the trade-off between grammar size and translation performance that is induced by including rules from multiple steps of the sampling process. It is helpful for translation quality to include more than one analysis of each sentence in the final grammar in order to increase coverage of new sentences. Averaging samples also better approximates the long-term behavior of the Markov chain, whereas taking a single sample involves an arbitrary random choice. When we average eight

Table 2
Comparisons of decoding results.

	iteration	model	pruning	#rules	dev	test	time (s)
Baseline		heuristic	Hiero	3.59M	25.5	25.1	809
No-singleton		heuristic	Hiero	1.09M	24.7	24.2	638
Sampled	0th (init)	Pitman-Yor	scope < 3	212K	19.9	19.1	489
Sampled	100th	Pitman-Yor	scope < 3	313K	23.9	23.3	1,214
Sampled	averaged (0 to 70)	Pitman-Yor	scope < 3	885K	26.2	24.5	1,488
Sampled	averaged (0 to 70)	Pitman-Yor	Hiero	785K	25.6	25.1	532
Sampled	averaged (0 to 70)	Dirichlet	scope < 3	774K	24.6	23.8	930

different samples, we get a larger number of rules than from a single sample, but still only a quarter as many rules as in the Hiero baseline. The translation results with eight samples are comparable to the Hiero baseline (not significantly different according to 1,000 iterations of paired bootstrap resampling [Koehn 2004]). Translation results are better with the sampled grammar than with the no-singleton method of reducing grammar size, while the sampled grammar was smaller than the no-singleton rule set. Thus, averaging samples seems to produce a good trade-off between grammar size and quality.

The filtering applied to the final rule set affects both the grammar size and decoding speed, because rules with different terminal/nonterminal patterns have varying decoding complexities. We experimented with two methods of filtering the final grammar: retaining rules of scope no greater than three, and the more restrictive the Hiero constraints. We do not see a consistent difference in translation quality between these methods, but there is a large impact in terms of speed. The Hiero constraints dramatically speeds decoding. The following is the full list of the Hiero constraints, taken verbatim from Chiang (2007):

- If there are multiple initial phrase pairs containing the same set of alignments, only the smallest is kept. That is, unaligned words are not allowed at the edges of phrases.
- Initial phrases are limited to a length of 10 words on either side.
- Rules are limited to five nonterminals plus terminals on the French side.
- Rules can have at most two nonterminals, which simplifies the decoder implementation. This also makes our grammar weakly equivalent to an inversion transduction grammar (Wu 1997), although the conversion would create a very large number of new nonterminal symbols.
- It is prohibited for nonterminals to be adjacent on the French side, a major cause of spurious ambiguity.
- A rule must have at least one pair of aligned words, so that translation decisions are always based on some lexical evidence.

Of these constraints, the differences between the Hiero constraints and scope filtering are: First, the Hiero constraints limit the number of nonterminals in a rule to no more than two. Second, the Hiero constraints do not allow two adjacent nonterminals in the source side of a rule. As discussed previously, these two differences limit Hiero grammar to be a subset of scope 2 grammar, whereas the scope-filtered grammar retains all scope 2 rules. Among grammars with the Hiero constraint, smaller grammars are generally faster. The relationship between the number of rules and the decoding time is less than linear. This is because the decoder never considers rules containing sequences of terminals not present in the source sentence. As the number of rules grows, we see rules with larger numbers of terminals that in turn apply to fewer input sentences. The sampled grammar has a more pronounced effect of reducing rule table size than decoding speed. Our sampling method may be particularly valuable for very large data sets where grammar size can become a limiting factor.

Finally, we wish to investigate whether the added power of the Pitman-Yor process gives any benefit over the simpler Dirichlet process prior, using the same modeling of word length in both cases. We find better translation quality with the Pitman-Yor

process, indicating that the additional strength of the Pitman-Yor process in suppressing infrequent rules helps prevent overfitting.

6. Conclusion

We presented a hypergraph sampling algorithm that overcomes the difficulties inherent in computing inside probabilities in applications where the segmentation of the tree into rules is not known.

Given parallel text with word-level alignments, we use this algorithm to learn sentence bracketing and SCFG rule composition. Our rule learning algorithm is based on a compact structure that represents all possible SCFG rules extracted from word-aligned sentences pairs, and works directly with highly lexicalized model parameters. We show that by effectively controlling overfitting with a Bayesian model, and designing algorithms that efficiently sample that parameter space, we are able to learn more compact grammars with competitive translation quality. Based on the framework we built in this work, it is possible to explore other rule learning possibilities that are known to help translation quality, such as learning refined nonterminals.

Our general sampling algorithm is likely to be useful in settings beyond machine translation. One interesting application would be unsupervised or partially supervised learning of (monolingual) TSGs, given text where the tree structure is completely or partially unknown, as in the approach of Blunsom and Cohn (2010b).

Acknowledgments

This work was partially funded by NSF grant IIS-0910611.

References

Blunsom, P., T. Cohn, C. Dyer, and M. Osborne. 2009. A Gibbs sampler for phrasal synchronous grammar induction. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2*, pages 782–790, Singapore.

Blunsom, Phil and Trevor Cohn. 2010a. Inducing synchronous grammars with slice sampling. In *Proceedings of the Human Language Technology: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 238–241, Boulder, CO.

Blunsom, Phil and Trevor Cohn. 2010b. Unsupervised induction of tree substitution grammars for dependency parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1,204–1,213, Cambridge, MA.

Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Chappelier, Jean-Cédric and Martin Rajman. 2000. Monte-Carlo sampling for NP-hard maximization problems in the framework of weighted parsing. In *Natural Language Processing (NLP 2000)*, pages 106–117, Patras.

Chiang, David. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Conference of the Association for Computational Linguistics (ACL-05)*, pages 263–270, Ann Arbor, MI.

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

Chiang, David, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Conference on Empirical Methods in Natural Language Processing (EMNLP-08)*, pages 224–233, Honolulu, HI.

Cohn, Trevor and Phil Blunsom. 2010. Blocked inference in Bayesian tree substitution grammars. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL-10)*, pages 225–230, Uppsala.

Cohn, Trevor, Sharon Goldwater, and Phil Blunsom. 2009. Inducing compact but accurate tree-substitution grammars. In *Proceedings of Human Language*

Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 548–556, Boulder, CO.

DeNero, John, Alexandre Bouchard-Cote, and Dan Klein. 2008. Sampling alignment structure under a Bayesian translation model. In *Conference on Empirical Methods in Natural Language Processing (EMNLP-08)*, pages 314–323, Honolulu, HI.

Galley, Michel, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the International Conference on Computational Linguistics/Association for Computational Linguistics (COLING/ ACL-06)*, pages 961–968, Sydney.

Galley, Michel, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of the 2004 Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-04)*, pages 273–280, Boston, MA.

Hopkins, Mark and Greg Langmead. 2010. SCFG decoding without binarization. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 646–655, Cambridge, MA.

Hopkins, Mark and Jonathan May. 2011. Tuning as ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1,352–1,362, Edinburgh.

Huang, Songfang and Steve Renals. 2010. Power law discounting for *n*-gram language models. In *Proceedings of the IEEE International Conference on Acoustic, Speech, and Signal Processing (ICASSP'10)*, pages 5,178–5,181, Dallas, TX.

Johnson, Mark, Thomas L. Griffiths, and Sharon Goldwater. 2007. Bayesian inference for PCFGs via Markov chain Monte Carlo. In *Proceedings of the 2007 Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-07)*, pages 139–146, Rochester, NY.

Knuth, D. E. 1975. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29(129):121–136.

Koehn, Philipp. 2004. Statistical significance tests for machine translation evaluation. In *2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 388–395, Barcelona.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-03)*, pages 48–54, Edmonton.

Levenberg, Abby, Chris Dyer, and Phil Blunsom. 2012. A Bayesian model for learning SCFGs with discontiguous rules. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 223–232, Jeju Island.

Liang, Percy and Dan Klein. 2009. Online EM for unsupervised models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 611–619, Boulder, CO.

Liu, Yang, Qun Liu, and Shouxun Lin. 2006. Tree-to-string alignment template for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 609–616, Sydney.

Naradowsky, Jason and Kristina Toutanova. 2011. Unsupervised bilingual morpheme segmentation and alignment with context-rich hidden semi-Markov models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 895–904, Portland, OR.

Neubig, Graham, Taro Watanabe, Eiichiro Sumita, Shinsuke Mori, and Tatsuya Kawahara. 2011. An unsupervised model for joint phrase alignment and extraction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 632–641, Portland, OR.

Och, Franz Josef. 2003. Minimum error rate training for statistical machine translation. In *Proceedings of the 41th Annual Conference of the Association for Computational Linguistics (ACL-03)*, pages 160–167, Sapporo.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*, pages 311–318, Philadelphia, PA.

Pitman, Jim and Marc Yor. 1997. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *Annals of Probability*, 25(2):855–900.

Post, Matt and Daniel Gildea. 2009. Bayesian learning of a tree substitution grammar. In *Proceedings of the Association for Computational Linguistics (short paper)*, pages 45–48, Singapore.

Sankaran, Baskaran, Gholamreza Haffari, and Anoop Sarkar. 2011. Bayesian extraction of minimal SCFG rules for hierarchical phrase-based translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 533–541, Edinburgh.

Teh, Yee Whye. 2006. A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 985–992, Sydney.

Vogel, Stephan, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 836–841, Copenhagen.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.

Zhang, Hao, Daniel Gildea, and David Chiang. 2008. Extracting synchronous grammar rules from word-level alignments in linear time. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING-08)*, pages 1,081–1,088, Manchester.

Zhang, Hao, Chris Quirk, Robert C. Moore, and Daniel Gildea. 2008. Bayesian learning of non-compositional phrases with synchronous parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL-08)*, pages 97–105, Columbus, OH.