Type-based MCMC for Sampling Tree Fragments from Forests

Xiaochang Peng and **Daniel Gildea**
Department of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

This paper applies type-based Markov Chain Monte Carlo (MCMC) algorithms to the problem of learning Synchronous Context-Free Grammar (SCFG) rules from a forest that represents all possible rules consistent with a fixed word alignment. While type-based MCMC has been shown to be effective in a number of NLP applications, our setting, where the tree structure of the sentence is itself a hidden variable, presents a number of challenges to type-based inference. We describe methods for defining variable types and efficiently indexing variables in order to overcome these challenges. These methods lead to improvements in both log likelihood and BLEU score in our experiments.

1 Introduction

In previous work, sampling methods have been used to learn Tree Substitution Grammar (TSG) rules from derivation trees (Post and Gildea, 2009; Cohn et al., 2009) for TSG learning. Here, at each node in the derivation tree, there is a binary variable indicating whether the node is internal to a TSG rule or is a split point, which we refer to as a **cut**, between two rules. The problem of extracting machine translation rules from word-aligned bitext is a similar problem in that we wish to automatically learn the best granularity for the rules with which to analyze each sentence. The problem of rule extraction is more complex, however, because the tree structure of the sentence is also unknown.

In machine translation applications, most previous work on joint alignment and rule extraction models uses heuristic methods to extract rules from learned word alignment or bracketing structures (Zhang et al., 2008; Blunsom et al., 2009; DeNero et al., 2008; Levenberg et al., 2012). Chung et al. (2014) present a MCMC algorithm schedule to learn Hiero-style SCFG rules (Chiang, 2007) by sampling tree fragments from phrase decomposition forests, which represent all possible rules that are consistent with a set of fixed word alignments. Assuming fixed word alignments reduces the complexity of the sampling problem, and has generally been effective in most state-of-the-art machine translation systems. The algorithm for sampling rules from a forest is as follows: from the root of the phrase decomposition forest, one samples a cut variable, denoting whether the current node is a cut, and an edge variable, denoting which incoming hyperedge is chosen, at each node of the current tree in a top-down manner. This sampling schedule is efficient in that it only samples the current tree and will not waste time on updating variables that are unlikely to be used in any tree.

As with many other token-based Gibbs Sampling applications, sampling one node at a time can result in slow mixing due to the strong coupling between variables. One general remedy is to sample blocks of coupled variables. Cohn and Blunsom (2010) and Yamangil and Shieber (2013) used blocked sampling algorithms that sample the whole tree structure associated with one sentence at a time for TSG and TAG learning. However, this kind of blocking does not deal with the coupling of variables correlated with the same type of structure across sentences. Liang et al. (2010) introduced a type-based sampling schedule which updates a block of variables of the same type jointly. The **type** of a variable is defined as the combination of new structural choices added when assigning different values to the variable. Type-based MCMC tackles the coupling issue by assigning the same type to variables that are strongly coupled.

In this paper, we follow the phrase decomposition forest construction procedures of Chung et

al. (2014) and present a type-based MCMC algorithm for sampling tree fragments from phrase decomposition forests which samples the variables of the same type jointly. We define the type of the cut variable for each node in our sampling schedule. While type-based MCMC has been proven to be effective in a number of NLP applications, our sample-edge, sample-cut setting is more complicated as our tree structure is unknown. We need additional steps to maintain the cut type information when the tree structure is changed as we sample the edge variable. Like other type-based MCMC applications, we need bookkeeping of node sites to be sampled in order to loop through sites of the same type efficiently. As noted by Liang et al. (2010), indexing by the complete type information is too expensive in some applications like TSG learning. Our setting is different from TSG learning in that the internal structure of each SCFG rule is abstracted away when deriving the rule type from the tree fragment sampled.

We make the following contributions:

1. We apply type-based MCMC to the setting of SCFG learning and have achieved better log likelihood and BLEU score result.

2. We present an innovative way of storing the type information by indexing on partial type information and then filtering the retrieved nodes according to the full type information, which enables efficient updates to maintain the type information while the amount of bookkeeping is reduced significantly.

3. We replace the two-stage sampling schedule of Liang et al. (2010) with a simpler and faster one-stage method.

4. We use parallel programming to do inexact type-based MCMC, which leads to a speed up of four times in comparison with non-parallel type-based MCMC, while the likelihood result of the Markov Chain does not change. This strategy should also work with other type-based MCMC applications.

2 MCMC for Sampling Tree Fragments from Forests

2.1 Phrase Decomposition Forest

The phrase decomposition forest provides a compact representation of all machine translation rules

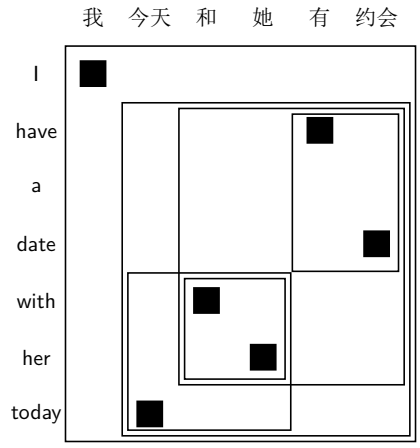

Figure 1: Example word alignment, with boxes showing valid phrase pairs. In this example, all individual alignment points are also valid phrase pairs.

that are consistent with our fixed input word alignment (Chung et al., 2014), and our sampling algorithm selects trees from this forest.

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 righthand side (r.h.s.), with at most two nonterminal occurrences on the r.h.s. Under this restriction, the maximum number of rules that can be extracted from an input sentence pair is $O(n^{12})$ with respect to the length of the sentence pair, as the left and right boundaries of the lefthand side (l.h.s.) nonterminal and each of the two r.h.s. nonterminals can take $O(n)$ positions in each of the two languages. This complexity leads us to explore sampling algorithms instead of using dynamic programming.

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 et al. (2003), i.e., 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

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

Each edge in E, written as $T \rightarrow n$, is made of a

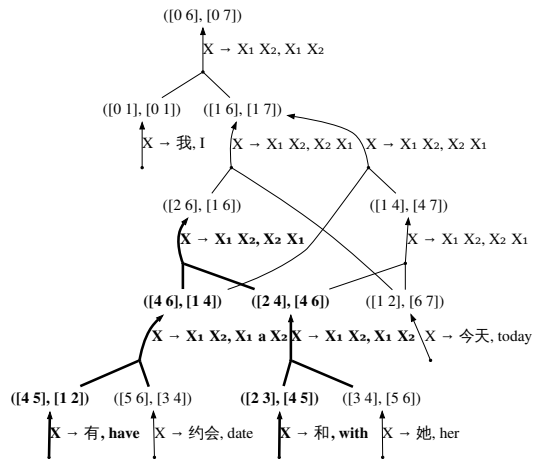

Figure 2: A phrase decomposition forest extracted from the sentence pair ⟨我 今天 和 她 有 约会, I have a date with her today⟩. Each edge is a minimal SCFG rule, and the rules at the bottom level are phrase pairs. Unaligned word "a" shows up in the rule $X \rightarrow X_1X_2, X_1aX_2$ after unaligned words are put back into the alignment matrix. The highlighted portion of the forest shows an SCFG rule built by composing minimal rules.

set of non-intersecting tail nodes $T \subset V$, and a single head node $n \in V$ that covers each tail node. We say 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 r.h.s. 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 1, the phrase decomposition forest is shown in Figure 2. Each boxed phrase in Figure 1 corresponds to a node in the forest of Figure 2, while hyperedges in Figure 2 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 2 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. Therefore, the problem of SCFG rules extraction can be solved by sampling tree fragments from the phrase decomposition forest. We use a bottom-up algorithm to construct the phrase decomposition forest from the word alignments.

2.2 Sampling Tree Fragments From Forest

We formulate the rule sampling procedure into two phases: first we select a tree from a forest, then we select the cuts in the tree to denote the split points between fragments, with each fragment corresponding to a SCFG rule. A tree can be specified by attaching a variable e_n to each node n in the forest, indicating which hyperedge is turned on at the current node. Thus each assignment will specify a unique tree by tracing the edge variables from the root down to the leaves. We also attach a cut variable z_n to each node, indicating whether the node is a split point between two adjacent fragments.

Let all the edge variables form the random vector Y and all the cut variables form the random vector Z. Given an assignment y to the edge variables and assignment z to the cut variables, our desired distribution is proportional to the product of weights of the rules specified by the assignment:

$$P_t(Y = y, Z = z) \propto \prod_{r \in \tau(y,z)} w(r) \qquad (1)$$

where $\tau(y, z)$ is the set of rules identified by the assignment. We use a generative model based on a Dirichlet Process (DP) defined over composed rules. We draw a distribution G over rules from a DP, and then rules from G.

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

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

$$P_0(r) = V_f^{-|r_f|} V_e^{-|r_e|} \qquad (2)$$

where V_f and V_e are the vocabulary sizes of the source language and the target language, and $|r_f|$ and $|r_e|$ are the lengths of the source side and target side of rule r. By marginalizing out G we get a simple posterior distribution over rules which can be described using the Chinese Restaurant Process (CRP). For this analogy, we imagine a restaurant has infinite number of tables that represent rule types and customers that represent translation rule instances. Each customer enters the restaurant and chooses a table to sit at. Let z_i be the table chosen by the i-th customer, then the customer chooses a table k either having been seated or a new table with probability:

$$P(z_i = k|z_{-i}) = \begin{cases} \frac{n_k}{i-1+\alpha} & 1 \leq k \leq K \\ \frac{\alpha}{i-1+\alpha} & k = K+1 \end{cases} \quad (3)$$

where z_{-i} is the current seating arrangement, n_k is the number of customers at the table k, K is the total number of occupied tables. If the customer sits at a new table, the new table will be assigned a rule label r with probability $P_0(r)$. We can see from Equation 3 that the only history related to the current table assignment is the counts in z_{-i}. Therefore, we define a table of counts $N = \{N_C\}_{C \in I}$ which memorizes different categories of counts in z_{-i}. I is an index set for different categories of counts. Each N_C is a vector of counts for category C. We have $P(r_i = r|z_{-i}) = P(r_i = r|N)$. If we marginalize over tables labeled with the same rule, we get the following probability over rule r given the previous count table N:

$$P(r_i = r|N) = \frac{N_R(r) + \alpha P_0(r)}{n + \alpha} \quad (4)$$

here in the case of DP, $I = \{R\}$, where R is the index for the category of rule counts. $N_R(r)$ is the number of times that rule r has been observed in z_{-i}, $n = \sum_r N_R(r)$ is the total number of rules observed.

We also define a Pitman-Yor Process (PYP) (Pitman and Yor, 1997) over rules of each length l. We draw the rule distribution G from a PYP, and then rules of length l are drawn from G.

$$G|\alpha, d, P_0 \sim PY(\alpha, d, P_0)$$

$$r|G \sim G$$

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. Integrating over G, we have the following PYP posterior probability:

$$P(r_i = r|N) = \frac{N_R(r) - T_r d + (T_l d + \alpha) P_0(r)}{N_L(l) + \alpha} \quad (5)$$

here for the case of PYP, $I = \{R, L\}$. We have an additional index L for the category of rule length counts, and $N_L(l)$ is the total number of rules of length l observed in z_{-i}. T_r is the number of tables labeled with r in z_{-i}. The length of the rule is drawn from a Poisson distribution, so a rule length probability $P(l; \lambda) = \frac{\lambda^l e^{-\lambda}}{l!}$ is multiplied by this probability to calculate the real posterior probability for each rule. In order to simplify the tedious book-keeping, we estimate the number of tables using the following equations (Huang and Renals, 2010):

$$T_r = N_R(r)^d \quad (6)$$

$$T_l = \sum_{r:|r|=l} N_R(r)^d \quad (7)$$

We use the *top-down* sampling algorithm of Chung et al. (2014) (see Algorithm 1). Starting from the root of the forest, we sample a value for the edge variable denoting which incoming hyperedge of the node is turned on in the current tree, and then we sample a cut value for the node denoting whether the node is a split point between two fragments in the tree. For each node n, we denote the composed rule type that we get when we set the cut of node n to 0 as r_1 and the two split rule types that we get when we set the cut to 1 as r_2, r_3. We sample the cut value z_i of the current node according to the posterior probability:

$$P(z_i = z|N) = \begin{cases} \frac{P(r_1|N)}{P(r_1|N)+P(r_2|N)P(r_3|N')} & \text{if } z = 0 \\ \frac{P(r_2|N)P(r_3|N')}{P(r_1|N)+P(r_2|N)P(r_3|N')} & \text{otherwise} \end{cases} \quad (8)$$

where the posterior probability $P(r_i|N)$ is according to either a DP or a PYP, and N, N' are tables of counts. In the case of DP, N, N' differ only in the rule counts of r_2, where $N'_R(r_2) = N_R(r_2) + 1$. In the case of PYP, there is an extra difference that

Algorithm 1 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
```

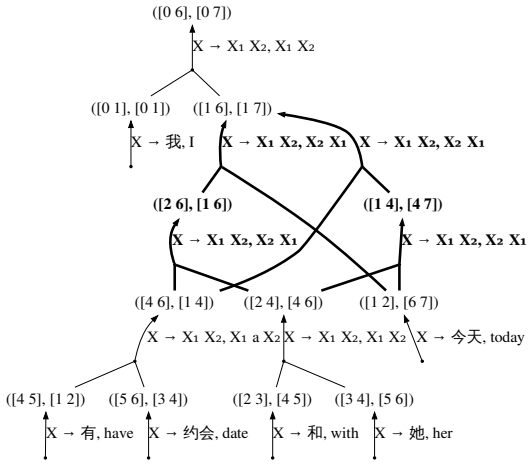

Figure 3: An example of cut type: Consider the two nodes marked in bold, $([2\,6],[1\,6])$, $([1\,4],[4\,7])$. These two non-split nodes are internal to the same composed rule: $X \rightarrow X_1X_2X_3, X_3X_2X_1$. We keep these two sites with the same index. However, when we set the cut value of these two nodes to 1, as the rules immediately above and immediately below are different for these two sites, they are not of the same type.

$N'_L(l) = N_L(l) + 1$, where l is the rule length of r_2.

As for edge variables e_i, we refer to the set of composed rules turned on below n including the composed rule fragments having n as an internal or root node as $\{r_1, \ldots, r_m\}$. We have the following posterior probability over the edge variable e_i:

$$P(e_i = e|N) \propto \prod_{i=1}^{m} P(r_i|N^{i-1}) \prod_{v \in \tau(e) \cap \text{in}(n)} \deg(v) \quad (9)$$

where $\deg(v)$ is the number of incoming edges for node v, $\text{in}(n)$ is the set of nodes in all subtrees under n, and $\tau(e)$ is the tree specified when we set $e_i = e$. N^0 to N^m are tables of counts where $N^0 = N$, $N^i_R(r_i) = N^{i-1}_R(r_i) + 1$ in the case of DP and additionally $N^i_L(l_i) = N^{i-1}_L(l_i) + 1$ in the case of PYP, where l_i is the rule length of r_i.

3 Type-based MCMC Sampling

Our goal in this paper is to organize blocks of variables that are strongly coupled into types and sample variables of each type jointly. One major property of type-based MCMC is that the joint probability of variables of the same type should be exchangeable so that the order of the variables does not matter. Also, the choices of the variables to be sampled jointly should not interfere with each other, which we define as a *conflict*. In this section, we define the type of cut variables in our sampling schedule and explain that with the two priors we introduced before, the joint probability of the variables will satisfy the exchangeability property. We will also discuss how to check conflict sites in our application.

In type-based MCMC, we need bookkeeping of sites as we need to loop through them to search for sites having the same type efficiently. In our two-stage sample-edge, sample-cut schedule, updating the edge variable would change the tree structure and trigger updates for the cut variable types in both the old and the new subtree. We come up with an efficient bookkeeping strategy to index on partial type information which significantly reduces the bookkeeping size, while updates are quite efficient when the tree structure is changed. The detail will become clear below.

3.1 Type-based MCMC

We refer to each node site to be sampled as a pair (t, n), indicating node n of forest t. For each site (t, n) and the corresponding composed rule types r_1 obtained when we set n's cut value to 0 and r_2, r_3 obtained when we set the cut value to 1, the cut variable type of site (t, n) is:

$$type(t, n) \stackrel{\text{def}}{=} (r_1, r_2, r_3)$$

We say that the cut variables of two sites are of the same type if the composed rule types r_1, r_2 and r_3 are exactly the same. For example, in Figure 3, assume that all the nodes in the hypergraph are currently set to be split points except for the two nodes marked in bold, $([2\,6],[1\,6]), ([1\,4],[4\,7])$. Considering these two non-split nodes, the composed rule types they are internal to (r_1) are exactly the same. However, the situation changes if we set the cut variables of these two nodes to be 1, i.e., all of the nodes in the hypergraph are now split points. As the rule type immediately above and the rule type immediately below the two nodes (r_2 and r_3) are now different, they are not of the same type.

We sample the cut value z_i according to Equation 8. As each rule is sampled according to a DP or PYP posterior and the joint probabilities according to both posteriors are exchangeable, we can see from Equation 8 that the joint prob-

ability of a sequence of cut variables is also exchangeable. Consider a set of sites S containing n cut variables $z_S = (z_1, ..., z_n)$ of the same type. This exchangeability property leads to the fact that any sequence containing same number of cuts (cut value of 1) would have same probability. We have the following probability distribution:

$$P(z_S|N) \propto \prod_{i=1}^{n-m} P(r_1|N^{i-1}) \prod_{i=1}^{m} P(r_2|\bar{N}^{i-1})P(r_3|\hat{N}^{i-1}) \stackrel{\text{def}}{=} g(m) \quad (10)$$

where N is the count table for all the other variables except for S. $m = \sum_{i=1}^{n} z_i$ is the number of cut sites. The variables N, $\bar{N}$, and $\hat{N}$ keep track of the counts as the derivation proceeds step by step:

$$\begin{aligned} N^0 &= N \\ N_R^i(r_1) &= N_R^{i-1}(r_1) + 1 \\ \bar{N}^0 &= N^{n-m} \\ \hat{N}_R^{i-1}(r_2) &= \bar{N}_R^{i-1}(r_2) + 1 \\ \bar{N}_R^i(r_3) &= \hat{N}_R^{i-1}(r_3) + 1 \end{aligned}$$

For PYP, we add extra count indices for rule length counts similarly.

Given the exchangeability property of the cut variables, we can calculate the posterior probability of $m = \sum_{i=1}^{n} z_i$ by summing over all $\binom{n}{m}$ combinations of the cut sites:

$$p(m|N) \propto \sum_{z_S: m=\sum_i z_i} p(z_S|N) = \binom{n}{m} g(m) \quad (11)$$

3.2 Sampling Cut-types

Given Equation 11 and the exchangeability property, our sampling strategy falls out naturally: first we sample m according to Equation 11, then conditioned on m, we pick m sites of z_S as cut sites out of the $\binom{n}{m}$ combinations with uniform probability.

Now we proceed to define **conflict** sites. In addition to exchangeability, another important property of type-based MCMC is that the type of each site to be sampled should be independent of the assignment of the other sites sampled at the same time. That is, in our case, setting the cut value of each site should not change the (r_1, r_2, r_3) triple of another site. We can see that the cut value of the current site would have effect on and only on the nodes in the r_1 fragment. We denote $nodes(r)$ as the node set for all nodes within fragment r. Then for $\forall z, z' \in S$, z is not in conflict with z' if and only if $nodes(r_1) \cap nodes(r_1') = \emptyset$, where r_1 and r_1' are the corresponding composed rule types when we set z, z' to 0.

Another crucial issue in type-based sampling is the bookkeeping of sampling sites, as we need to loop through all sites having the same type with the current node. We only maintain the type information of nodes that are currently turned on in the chosen tree of the forest, as we only sample these nodes. It is common practice to directly use the type value of each variable as an index and maintain a set of sites for each type. However, maintaining a (r_1, r_2, r_3) triple for each node in the chosen tree is too memory heavy in our appli-

Algorithm 2 Type-based MCMC Algorithm for Sampling One Site

```
sample one type of sites, currently sample site (node, parent)
if parent is None or node is sampled then
    return
end if
old = node.cut
node.cut = 0
r_1 = composed_rule(parent)
node.cut = 1
r_2 = composed_rule(parent)
r_3 = composed_rule(node)
node.cut = old
sites =
for sites s ∈ index[r_1] do
    for sites s' in rule rooted at s do
        if s' of type (r_1, r_2, r_3) and no conflict then
            add s' to sites
        end if
    end for
end for
for sites s ∈ index[r_3] do
    if s of type (r_1, r_2, r_3) and no conflict then
        add s to sites
    end if
end for
sample m according to Equation 11
remove sites from index
uniformly choose m in sites to be cut sites.
add new cut sites to index
mark all nodes in sites as sampled
```

cation.

In our two-stage sample-edge, sample-cut schedule, there is an additional issue that we must deal with efficiently: when we have chosen a new incoming edge for the current node, we also have to update the bookkeeping index as the current tree structure is changed. Cut variable types in the old subtree will be turned off and a new subtree of variable types will be turned on. In the extreme case, when we have chosen a new incoming edge at the root node, we have chosen a new tree in the forest. So, we need to remove appearances of cut variable types in the old tree and add all cut variable types in the newly chosen tree.

Our strategy to deal with these two issues is to build a small, simple index, at the cost of some additional computation when retrieving nodes of a specified type. To be precise, we build an index from (single) rule types r to all occurrences of r in the data, where each occurrence is represented as a pointer to the root of r in the forest. Our strategy has two important differences from the standard strategy of building an index having the complete type (r_1, r_2, r_3) as the key and having every node as an entry. Specifically:

1. We index only the roots of the current rules, rather than every node, and

2. We key on a single rule type, rather than a triple of rule types.

Differences (1) and (2) both serve to keep the index small, and the dramatic savings in memory is essential to making our algorithm practical. Furthermore, difference (1) reduces the amount of work that needs to be done when an edge variable is resampled. While we must still re-index the entire subtree under the changed edge variable, we need only to re-index the roots of the current tree fragments, rather than all nodes in the subtree.

Given this indexing strategy, we now proceed to describe the process for retrieving nodes of a specified type (r_1, r_2, r_3). These nodes fall into one of two cases:

1. Internal nodes, i.e., nodes whose cut variable is currently set to 0. These nodes must be contained in a fragment of rule type r_1, and must furthermore have r_2 above them, and r_3 below them. We retrieve these nodes by looking up r_1 in the index, iterating over all nodes in each fragment retrieved, and retaining only those with r_2 above and r_3 below. (Lines 13–19 in Algorithm 2.)

2. Boundary nodes, i.e., nodes whose cut variable is currently set to 1. These nodes must form the root of a fragment r_3, and have a fragment r_2 above them. We retrieve these nodes by looking up r_3 in the index, and then checking each node retrieved to retain only those nodes with r_2 above them in the current tree. (Lines 20–24 in Algorithm 2.)

This process of winnowing down the nodes retrieved by the index adds some computational overhead to our algorithm, but we find that it is minimal in practice.

We still use the top-down sampling schedule of Algorithm 1, except that in the *sample-edge* step, when we choose a new incoming edge, we add additional steps to update the bookkeeping index. Furthermore, in the *sample-cut* step, we sample all non-conflict sites having the same type with n jointly. Our full algorithm for sampling one cut-type is shown in Algorithm 2. When sampling each site, we record a $parent$ node of the nearest cut ancestor of the current node so that we can build r_1 and r_2 more quickly, as they are both rooted at $parent$. We first identify the type of the current site. Then we search the bookkeeping index to find possible candidate sites of the same type, as described above. As for conflict checking, we keep a set of nodes that includes all nodes in the r_1 fragment of previous non-conflict sites. If the r_1 fragment of the current site has any node in common with this node set, we arrive at a conflict site.

4 Methods of Further Optimization

4.1 One-stage Sampling Schedule

Instead of calculating the posterior of each m according to Equation 11 and then sampling m, we can build our real m more greedily.

$$P(z_S|N) = \prod_{i=1}^{n} P(z_i|N^{i-1}) \qquad (12)$$

where $N, N^0, \ldots, N^n$ are count tables, and $N^0 = N$. N^i is the new count table after we update N^{i-1} according to the assignment of z_i. This equation gives us a hint to sample each z_i according to $P(z_i|N^{i-1})$ and then update the count table N^{i-1} according to the assignment of z_i. This greedy

sampling saves us the effort to calculate each m by multiplying over each posterior of cut variables but directly samples the real m. In our experiment, this one-stage sampling strategy gives us a 1.5 times overall speed up in comparison with the two-stage sampling schedule of Liang et al. (2010).

4.2 Parallel Implementation

As our type-based sampler involves tedious bookkeeping and frequent conflict checking and mismatch of cut types, one iteration of the type-based sampler is slower than an iteration of the token-based sampler when run on a single processor. In order to speed up our sampling procedure, we used a parallel sampling strategy similar to that of Blunsom et al. (2009) and Feng and Cohn (2013), who use multiple processors to perform inexact Gibbs Sampling, and find equivalent performance in comparison with an exact Gibbs Sampler with significant speed up. In our application, we split the data into several subsets and assign each subset to a processor. Each processor performs type-based sampling on its subset using local counts and local bookkeeping, and communicates the update of the local counts after each iteration. All the updates are then aggregated to generate global counts and then we refresh the local counts of each processor. We do not communicate the update on the bookkeeping of each processor. In this implementation, we have a slightly "out-of-date" counts at each processor and a smaller bookkeeping of sites of the same type, but we can perform type-based sampling independently on each processor. Our experiments show that, with proper division of the dataset, the final performance does not change, while the speed up is significant.

5 Experiments

We used the same LDC Chinese-English parallel corpus as Chung et al. (2014),[1] which is composed of newswire text. The corpus consists of 41K sentence pairs, which has 1M words on the English side. The corpus has 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. We plotted the log likelihood graph to compare the convergence property of each sampling schedule and calculated BLEU (Papineni et al., 2002) for evaluation.

5.1 Experiment Settings

We use the top-down token-based sampling algorithm of Chung et al. (2014) as our baseline. We use the same SCFG decoder for translation with both the baseline and the grammars sampled using our type-based MCMC sampler. The features included in our experiments are differently normalized rule counts and lexical weightings (Koehn et al., 2003) of each rule. Weights are tuned using Pairwise Ranking Optimization (Hopkins and May, 2011) using a grammar extracted by the standard heuristic method (Chiang, 2007) and the development set. The same weights are used throughout our experiments.

First we want to compare the DP likelihood of the baseline with our type-based MCMC sampler to see if type-based sampling would converge to a better sampling result. In order to verify if type-based MCMC really converges to a good optimum point, we use simulated annealing (Kirkpatrick et al., 1983) to search possible better optimum points. We sample from the real distribution modified by an annealing parameter β:

$$z \sim P(z)^{\beta}$$

We increase our β from 0.1 to 1.3, and then decrease from 1.3 to 1.0, changing by 0.1 every 3 iterations. We also run an inexact parallel approximation of type-based MCMC in comparison with the non-parallel sampling to find out if parallel programming is feasible to speed up type-based MCMC sampling without affecting the performance greatly. We do not compare the PYP likelihood because the approximation renders it impossible to calculate the real PYP likelihood. We also calculate the BLEU score to compare the grammars extracted using each sampling schedule. We just report the BLEU result of grammars sampled using PYP as for all our schedules, since PYP always performs better than DP.

[1] The data are randomly sampled 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.

As for parameter settings, we used $d = 0.5$ for the Pitman-Yor discount parameter. Though we have a separate PYP for each rule length, we used same $\alpha = 5$ for all rule sizes in all experiments, including experiments using DP. For rule length probability, a Poisson distribution where $\lambda = 2$ was used for all experiments.[3]

For each sentence sample, we initialize all the nodes in the forest to be cut sites and choose an incoming edge for each node uniformly. For each experiment, we run for 160 iterations. For each DP experiment, we draw the log likelihood graph for each sampling schedule before it finally converges. For each PYP experiment, we tried averaging the grammars from every 10th iteration to construct a single grammar and use this grammar for decoding. We tune the number of grammars included for averaging by comparing the BLEU score on the dev set and report the BLEU score result on the test with the same averaging of grammars.

As each tree fragment sampled from the forest represents a unique translation rule, we do not need to explicitly extract the rules; we merely need to collect them and count them. However, the fragments sampled include purely non-lexical rules that do not conform to the rule constraints of Hiero, and rules that are not useful for translation. In order to get rid of this type of rule, we prune every rule that has **scope** (Hopkins and Langmead, 2010) greater than two. Whereas Hiero does not allow two adjacent nonterminals in the source side, our pruning criterion allows some rules of scope two that are not allowed by Hiero. For example, the following rule (only source side shown) has scope two but is not allowed by Hiero:

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

5.2 Experiment Results

Figure 4 shows the log likelihood result of our type-based MCMC sampling schedule and the baseline top-down sampling. We can see that type-based sampling converges to a much better result than non-type-based top-down sampling. This shows that type-based MCMC escapes some local optima that are hard for token-based methods to escape. This further strengthens the idea that sampling a block of strongly coupled variables jointly

[3]The priors are the same as the work of Chung et al. (2014). The priors are set to be the same because other priors turn out not to affect much of the final performance and add additional difficulty for tuning.

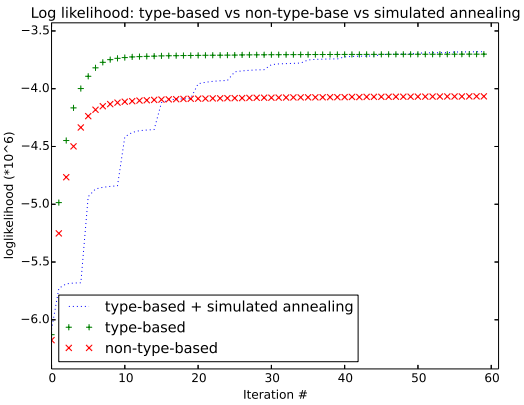

Figure 4: Log likelihood result of type-based MCMC sampling against non-type-based MCMC sampling, simulated annealing is used to verify if type-based MCMC converges to a good likelihood

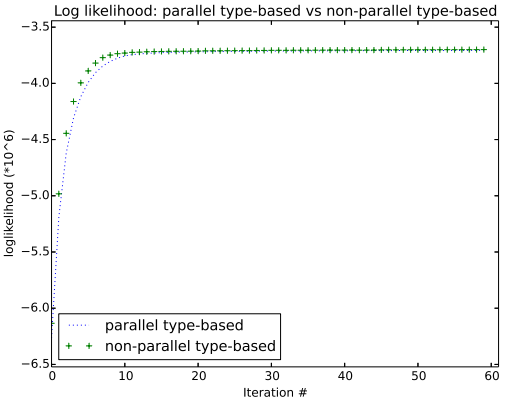

Figure 5: parallelization result for type-based MCMC

helps solve the slow mixing problem of token-based sampling methods. Another interesting observation is that, even though theoretically these two sampling methods should finally converge to the same point, in practice a worse sampling algorithm is prone to get trapped at local optima, and it will be hard for its Markov chain to escape it. We can also see from Figure 4 that the log likelihood result only improves slightly using simulated annealing. One possible explanation is that the Markov chain has already converged to a very good optimum point with type-based sampling and it is hard to search for a better optimum.

Figure 5 shows the parallelization result of type-based MCMC sampling when we run the program on five processors. We can see from the graph that when running on five processors, the likelihood fi-

Sampling Schedule	iteration	dev	test
Non-type-based	averaged (0-90)	25.62	24.98
Type-based	averaged (0-100)	25.88	25.20
Parallel Type-based	averaged (0-90)	25.75	25.04

Table 1: Comparisons of BLEU score results

nally converges to the same likelihood result as non-parallel type-based MCMC sampling. However, when we use more processors, the likelihood eventually becomes lower than with non-parallel sampling. This is because when we increase the number of processors, we split the dataset into very small subsets. As we maintain the bookkeeping for each subset separately and do not communicate the updates to each subset, the power of type-based sampling is weakened with bookkeeping for very few sites of each type. In the extreme case, when we use too many processors in parallel, the bookkeeping would have a singleton site for each type. In this case, the approximation would degrade to the scenario of approximating token-based sampling. By choosing a proper size of division of the dataset and by maintaining local bookkeeping for each subset, the parallel approximation can converge to almost the same point as non-parallel sampling. As shown in our experimental results, the speed up is very significant with the running time decreasing from thirty minutes per iteration to just seven minutes when running on five processors. Part of the speed up comes from the smaller bookkeeping since with fewer sites for each index, there is less mismatch or conflict of sites.

Table 1 shows the BLEU score results for type-based MCMC and the baseline. For non-type-based top-down sampling, the best BLEU score result on dev is achieved when averaging the grammars of every 10th iteration from the 0th to the 90th iteration, while our type-based method gets the best result by averaging over every 10th iteration from the 0th to the 100th iteration. We can see that the BLEU score on dev for type-based MCMC and the corresponding BLEU score on test are both better than the result for the non-type-based method, though not significantly. This shows that the better likelihood of our Markov Chain using type-based MCMC does result in better translation.

We have also done experiments calculating the BLEU score result of the inexact parallel implementation. We can see from Table 1 that, while the likelihood of the approximation does not change in comparison with the exact type-based MCMC, there is a gap between the BLEU score results. We think this difference might come from the inconsistency of the grammars sampled by each processor within each iteration, as they do not communicate the update within each iteration.

6 Conclusion

We presented a novel type-based MCMC algorithm for sampling tree fragments from phrase decomposition forests. While the hidden tree structure in our settings makes it difficult to maintain the constantly changing type information, we have come up with a compact way to store the type information of variables and proposed efficient ways to update the bookkeeping index. Under the additional hidden structure limitation, we have shown that type-based MCMC sampling still works and results in both better likelihood and BLEU score. We also came with techniques to speed up the type-based MCMC sampling schedule while not affecting the final sampling likelihood result. A remaining issue with parallelization is the inconsistency of the grammar within an iteration between processors. One possible solution would be using better averaging methods instead of simply averaging over every few iterations. Another interesting extension for our methods would be to also define types for the edge variables, and then sample both cut and edge types jointly.

Acknowledgments

We gratefully acknowledge the assistance of Licheng Fang and Tagyoung Chung. This work was partially funded by NSF grant IIS-0910611.

References

Phil Blunsom, Trevor Cohn, Chris Dyer, and Miles 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.

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

Tagyoung Chung, Licheng Fang, Daniel Gildea, and Daniel Štefankovič. 2014. Sampling tree fragments from forests. *Computational Linguistics*, 40:203–229.

Trevor Cohn 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, Sweden.

Trevor Cohn, 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, Colorado, June. Association for Computational Linguistics.

John DeNero, Alexandre Bouchard-Cote, and Dan Klein. 2008. Sampling alignment structure under a Bayesian translation model. In *Proceedings of EMNLP*, pages 314–323, Honolulu, HI.

Yang Feng and Trevor Cohn. 2013. A markov model of machine translation using non-parametric bayesian inference. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 333–342, Sofia, Bulgaria, August. Association for Computational Linguistics.

Mark Hopkins 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, October.

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362, Edinburgh, Scotland, UK., July.

Songfang Huang and Steve Renals. 2010. Power law discounting for n-gram language models. In *Proc. IEEE International Conference on Acoustic, Speech, and Signal Processing (ICASSP'10)*, pages 5178–5181, Dallas, Texas, USA.

Scott Kirkpatrick, C. D. Gelatt, Jr., and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of NAACL-03*, pages 48–54, Edmonton, Alberta.

Abby Levenberg, 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, Korea.

Percy Liang, Michael I Jordan, and Dan Klein. 2010. Type-based mcmc. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 573–581. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of ACL-02*, pages 311–318, Philadelphia, PA.

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

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

Elif Yamangil and Stuart M Shieber. 2013. Non-parametric bayesian inference and efficient parsing for tree-adjoining grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. Association of Computational Linguistics.

Hao Zhang, Daniel Gildea, and David Chiang. 2008. Extracting synchronous grammar rules from word-level alignments in linear time. In *COLING-08*, pages 1081–1088, Manchester, UK.