Sampling from Probabilistic Context-Free Grammars

Iason Sebastiaan de Bondt
10266224

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr Wilker Ferreira Aziz

Institute for Logic, Language and Computation
Faculty of Science
University of Amsterdam
Room F2.11
Building F
Science Park 107
1098 XG Amsterdam

June 26th, 2015

Abstract

In this present study there are two sampling methods, applicable to PCFGs, implemented and compared with respect to each other, i.e. ancestral sampling (a form of Monte-Carlo sampling), and slice sampling (a form of Markov chain Monte-Carlo) sampling. Ancestral sampling requires the exact representation of the target probability density function, which can be computed with a chart-based parsing algorithm (e.g. CKY). However, due to the cubic complexity of such algorithms, these approaches become impracticable when grammars or input sentences become too large. In this study it is argued that a complete run of a chart-based parsing algorithm is wasteful, in the sense that most of the items enumerated by these algorithms are unlikely to be sampled. Slice sampling is an alternative to regular MC sampling approaches which reasons over high probability regions of the target distribution rather than over the whole distribution. In this thesis it is investigated whether slice sampling reduces the complexity of sampling PCFGs with respect to ancestral sampling, in addition, if slice sampling could preserve the accuracy of ancestral sampling. The results are particularly encouraging. The slice sampling approach reduced the complexity for both the sampling and the parsing time, with respect to ancestral sampling. Furthermore, the accuracy of slice sampling is really close to the exact one. This study showed that slice sampling is an excellent alternative to MC sampling approach when grammars or input sentences become too large.

Contents

1 Introduction

The field of Natural Language Processing (NLP) is concerned with natural language understanding and generation. More broadly, NLP is about enabling computers to deal with human languages. Many NLP tasks involve natural language understanding. For that purpose, language can be analysed in terms of smaller, compositional building blocks. *Context-free grammars* (CFGs) offer a formal treatment of such hierarchical representations with convenient and well-understood properties (Hopcroft and Ullman, 1969). Moreover, many linguistic formalisms of language can be efficiently represented by CFGs (Kallmeyer, 2010). Applications of CFGs to NLP include syntactic parsing (Klein and Manning, 2003) and machine translation (Chiang, 2005). For example, CFGs enable the possibility to parse sentences and represent their syntactic structure as constituent trees; these representations are called *parse trees*. Many CFGs are inherently ambiguous, which is particularly true for those estimated from data. To quantify this ambiguity, or to characterise this degree of uncertainty, CFGs can be made probabilistic. This probabilistic extension, a PCFG, allows one to assess the likelihood of a parse tree as well as the likelihood of a sentence as a whole.

In most cases, only one parse tree out of the distribution defined by a PCFG is selected, which can be seen as a form of statistical disambiguation. In many of those cases, as a modelling convenience, PCFGs are augmented with latent variables (Petrov et al., 2006a). Under such latent-variable PCFGs (LPCFGs), a derivation corresponds to a parse tree paired with a specific assignment of the latent variables. Such derivations can be thought of as latent refinements (or specialisations) of the underlying parse tree. This is a common artefact in parsing and in machine translation, where latent variables represent phrase category refinements and bilingual phrase alignments, respectively.

Determining the most probable parse (MPP) of a sentence under a PCFG is a very common task within NLP. In the particular case of the latent PCFG, this maximization task corresponds to an NP-hard[1] problem (Sima'an, 1996). Sampling methods offer a tractable and sound alternative to dealing with the MPP problem. In a nutshell, a reduced though representative statistical sample of derivations is used to reason about underlying parse trees. Another typical disambiguation criterion (common to standard PCFGs and those with latent variables) is the minimum Bayes risk (MBR) principle (Sima'an, 2003), whereby the parse tree that minimises an expected loss function is selected. This criterion can be rather complex depending on the nature of the loss function. In such cases, it is natural to rely on sampling in order to compute expectations with respect to a statistical sample of derivations as opposed to the entire distribution.

Monte-Carlo (MC) methods (Robert and Casella, 2004) are techniques for approximation of complex probability distributions through random sampling. MC sampling typically requires an unpruned representation of the target *probability density function* (PDF). In the case of PCFGs, this representation consists of a weighted forest of derivations (also known as a chart). Such forests can be produced in polynomial time by dynamic programming algorithms such as CKY

[1]Non-deterministic Polynomial-time hard.

(Hopcroft and Ullman, 1969) and Earley parser (Earley, 1970).

Chappelier and Rajman (2000) introduced a sampling technique which is applicable to PCFGs based on the principle of *ancestral sampling*, a form of unbiased sampling which produces independent samples from the distribution of derivations. They first perform a complete (unpruned) run of CKY in order to represent the target distribution exactly. Subsequently, they perform a linear-time traversal of the forest drawing random samples directly from the inverted *cumulative density function* (CDF). To obtain the exact CDF, they use the Inside algorithm (Baker, 1979), a dynamic programming procedure which runs in linear time with the size of the forest.

However, the use of a parsing algorithm that derives all possible derivations might be naïve for the purpose of sampling in the sense that most of the items enumerated by such an algorithm are unlikely to be sampled resulting in an unnecessary complexity of $\mathcal{O}(G^3 n^3)$, where G and n are the size of the grammar and input sentence respectively. Consequently, the use of such an algorithm causes the MC sampling approach to be inefficient, which makes it impracticable when grammars or sentences become too large. Certainly, this is not a problem for small grammars made by human specialists, however, this approach may become impracticable for larger grammars, for instance, grammars that are automatically learned from data. An example of such a grammar is given by Slav Petrov et al., in which an automatic approach to tree annotation is presented (Petrov et al., 2006b).

State-of-the-art alternatives have been proposed, whereby sampling can be accomplished without a complete run of a chart-based parsing algorithm (e.g. CKY). These alternatives are known as Markov chain Monte-Carlo (MCMC) sampling approaches. MCMC approaches sample from a probability distribution based on a constructed Markov chain, which leaves the target distribution invariant. They are iterative procedures that resample blocks of variables while leaving the remaining variables unchanged. Particularly in the case of PCFGs, a derivation is sampled from a reduced set of neighbouring derivations to a given previously sampled solution.

Slice sampling (Neal, 2003) is one example of an MCMC technique which is compatible with many applications for it requires little knowledge about the specificity of the target distribution. Particularly, in the case of PCFGs, slice sampling reduces the dimensionality of the problem by 'slicing' the distribution over derivations into multiple (possibly overlapping) segments. These segments contain the derivations that are most likely to occur at a given time. This strategy greatly reduces the complexity of parsing. Blunsom and Cohn (2010) derived a slice sampler for PCFGs with applications to machine translation.

This research involves the implementation of slice sampling for PCFGs, as an alternative to regular MC sampling. The following two questions will be addressed:

Q1 *To what extent can slice sampling reduce the complexity of sampling PCFGs in comparison to MC sampling?*

Q2 *To what extent will slice sampling preserve the accuracy of MC sampling?*

The hypothesis is that a complete run of a chart-based parsing algorithm (e.g CKY and Earley) is wasteful, in the sense that most of the items enumerated by these algorithms are unlikely to be sampled. The expected result of this research is an implementation of slice sampling as an alternative to regular MC sampling. Ideally, this will enable the possibility to handle larger grammars such as those that naturally arise from data-driven approaches to parsing and machine translation (Petrov et al., 2006b). On the other hand, MCMC techniques, such as slice sampling, produce autocorrelated samples, which potentially requires a larger number of samples before reliable estimates and decisions can be made.

This thesis is divided into five chapters. In the second chapter, a comprehensive description is provided of the theory that is necessary to understand the conducted research. The third chapter is dedicated to the elaboration of the utilized research method. The results of this research are provided and evaluated in the fourth chapter. Finally, a conclusion will be drawn that will attempt to answer the research questions.

2 Theoretical Framework

This chapter consists of a comprehensive description of the theory that is necessary to understand the conducted research. Initially, some background information of NLP is provided, including the definitions of context-free grammars, parsers and probabilistic parsing algorithms. Thereafter, the intuitions of sampling from the inverted CDF and slice sampling are elaborated.

2.1 Background Information

This section consists of some background information of NLP that is necessary to understand the succeeding part of this chapter. Firstly, a definition is given of the language formalisms used in this research. Secondly, the intuition of parsing in general is elaborated along with the explanation of some parsing algorithms. Finally, an explanation is provided of probabilistic parsing as well as the algorithms that are required to achieve it.

2.1.1 Context-Free Grammars

In order to process natural language, a formalization of the grammar rules is required; this is often accomplished with a *context-free grammar* (CFG). A CFG is a four-tuple (N, Σ, R, S) where N is the set of non-terminals (e.g. verb or noun phrase) including the start symbol S, Σ is the set of terminal symbols (i.e. words), and R is the set of rules, each of the form $A \to \alpha$, where A is a non-terminal and α is a sequence of terminals or non-terminals, i.e. $A \in N$ and $\alpha \in (N \cup \Sigma)^*$ (Hopcroft and Ullman, 1969). CFGs enable the possibility to parse sentences and represent their syntactic structure as constituent trees; these representations are called *parse trees*, or more generally, *derivations*. Formally, a derivation is a sequence of rule applications starting from the start symbol S, rewriting every non-terminal, yielding a terminal string. An example derivation of the sentence ‘the cat drinks’ according to an example CFG is given in figure 2.

The central assumption regarding CFGs is that non-terminal symbols are rewritten with no dependency on its surrounding context. That is precisely the implication of rules of the form

$A \to \alpha$, i.e. all that needs to be known is the identity of the non-terminal symbol, its position and context in a derivation do not matter.

A probabilistic context-free grammar (PCFG) is an extension of a CFG, where probabilities are assigned to each rule. A central assumption regarding PCFGs is that rewrite rules are independent of one another conditioned on the non-terminal symbol they rewrite. That is, if $A \to \alpha$ and $A \to \beta$ are rules in the grammar, then α and β are independent events given A. This lead to rules being assigned probabilities of the form $P(A \to \alpha) \equiv P(\alpha \mid A)$ such that $0 \leq P(A \to \alpha) \leq 1$ and $\Sigma_\alpha P(A \to \alpha) = 1$ for every $A \in N$ and $\alpha \in (N \cup \Sigma)^*$. Given this conditional dependence assumption, the probability of a derivation d factorises as a product over rule probabilities as shown in equation (1).

$$P(x, d) = \prod_{A \to \alpha \in d} P(\alpha \mid A) \tag{1}$$

Formally, this is the joint probability of the string $x \in \Sigma^*$ and a derivation d. The most probable derivation of a sentence is the one that maximizes $P(x, d)$ out of the set of all derivations $D(x) = \{d : S \overset{d}{\Rightarrow} x\}$, where the notation $S \overset{d}{\Rightarrow} x$ means that x can be derived from the start symbol S through d.

$$d^* = \operatorname*{argmax}_{d \in D(x)} P(x, d) \tag{2}$$

The total probability of a sentence follows by marginalization, i.e. the summation over all analyses of the sentence:

$$P(x) \;=\; \sum_{d \in D(x)} P(x, d) \;=\; \sum_{d \in D(x)} \prod_{r \in d} P(r) \tag{3}$$

Where r represents $A \to \alpha$ and $P(r)$ stands for $P(\alpha \mid A)$.

2.1.2 Parsers

Natural languages are often ambiguous, consequently, often there are numerous derivations per sentence. Parsing is the process of determining the phrase structure (derivation) of a sentence, complying to the rules of a formal grammar, e.g. a CFG. A parsing algorithm computes all possible parse tree derivations of a sentence. There exists several parsing algorithms, distinguishable in *top-down* and *bottom-up* parsers. This section will proceed by presenting two parsing algorithms.

The CKY algorithm (Hopcroft and Ullman, 1969), is a bottom-up, dynamic parsing algorithm applicable to CFGs. The complexity of the CKY algorithm is $\mathcal{O}(G^3 n^3)$, where G is the size of grammar and n the size of the parsed sentence. It starts with the terminal symbols (i.e. words) and attempts to complete rules recursively, until the rule containing the goal (i.e. the root) item has been derived. This is often realized by a bottom-up filling of a parse chart with items that represent all possible sub-derivations of all possible substrings of the input sentence. Figure 1a illustrates, in terms of deduction rules[2], a generalized version of CKY presented by Nederhof and Satta (2008), which deals with an arbitrary epsilon-free CFG.

Goal item: [S,0,n]

Axioms: $$\frac{}{[A \to \bullet\alpha, i, i]} \quad \begin{matrix} A \to \alpha \in G \\ 0 \leq i \leq n \end{matrix}$$

Scan: $$\frac{[A \to \alpha \bullet a\beta, i, j]}{[A \to \alpha a \bullet \beta, i, j+1]} \quad w_{j+1} = a$$

Complete: $$\frac{[A \to \alpha \bullet B\beta, i, j], [B \to \gamma\bullet, j, k]}{[A \to \alpha B_{j-k} \bullet \beta, i, k]}$$

(a) Nederhof et al.

Goal items: $[S \to \alpha\bullet, 0, n]$ $\quad S \to \alpha \in P$

Axioms: $$\frac{}{[S \to \bullet\alpha, 0, 0]} \quad S \to \alpha \in P$$

Predict: $$\frac{[A \to \alpha \bullet B\beta, i, j]}{[B \to \bullet\gamma, j, j]} \quad B \to \gamma \in P$$

Scan: $$\frac{[A \to \alpha \bullet a\beta, i, j]}{[A \to \alpha a \bullet \beta, i, j+1]} \quad w_{j+1} = a$$

Complete: $$\frac{[A \to \alpha \bullet B\beta, i, j], [B \to \gamma\bullet, j, k]}{[A \to \alpha B_{j-k} \bullet \beta, i, k]}$$

(b) Earley.

Figure 1: The generalised CKY (a) and the Earley (b) algorithms for parsing (P)CFGs.

The Earley algorithm (Earley, 1970) is often described as a bottom-up parser with top-down filtering (Goodman, 1999); nodes are predicted in a top-down fashion, consequently, filtering nodes that cannot be derived, additionally, nodes are bottom-up completed. An item-based description of the Earley parser is illustrated in figure 1b. Generally, items have the form: $A \to \alpha \bullet \beta, i, j$, with $A \in N$ and $\alpha, \beta \in (N \cup \Sigma)$ and $0 \leq i \leq j \leq n$, where n is the length of the input sentence. The dot symbol indicates the status of the nodes, a dot that precedes or follows a node, indicates a predicted or a completed node respectively. The symbols i and j are the start and end positions respectively, indicating the span of the completed part of the input sentence.

Formally, the Earley algorithm consists of four operations: *axioms*, *predict*, *scan* and *complete*. It initiates with the axioms, where rules that contain the root item (indicated with an S) in their left-hand side are derived. Symbols in the right hand side of the derived rules are progressed one at the time, by moving the dot over the adjacent symbol. One of the remaining three operations is executed, depending whether the symbol at the right side of the dot is a terminal or a non-terminal symbol, or the dot has reached the end of the right-hand sided sequence.

The predict operation is applicable when there is a non-terminal at the right side of the dot. Every rule in the grammar that contains the current non-terminal symbol as its left-hand side is derived. Additionally, these derived rules will be progressed as well, recursively.

[2]The general form of deduction rules is as follows:

$$\frac{\textit{antecedent}}{\textit{consequent}} \quad \textit{side conditions}$$

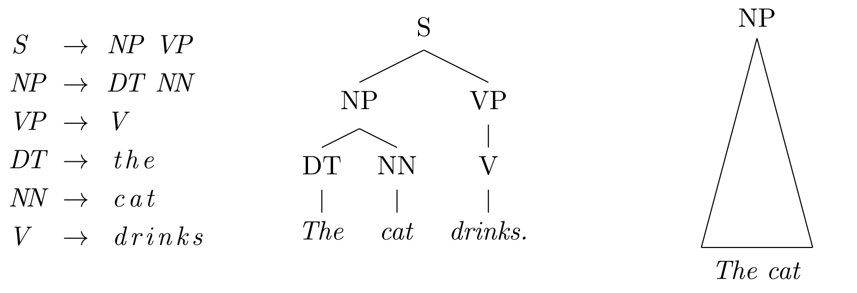
The antecedent and consequent are lists of items, if the antecedent can be deduced and the side conditions hold, then, the consequent items can be deduced as well (Shieber et al., 1995).

The scan operation is applicable whenever the right side of the dot is occupied by a terminal symbol. The terminal symbol will be compared with the subsequent word in the input sentence, the operation is successful when they match. Consequently, a successful scan operation will complete the terminal node, and otherwise, if the operation fails, discard the rule associated with the terminal symbol.

The complete operation is applicable whenever the dot is at the end of its production, implying that the current item is complete. Consequently, completing previously deduced items. The sentence is parseable if a goal item is deduced.

The program runs exhaustively deducing new items from previously deduced ones. It can be easily implemented by iterating over an agenda of active items which are processed one at the time. Active items combine with passive ones (those that have left the active agenda) through the operations (e.g. predict, complete, scan) potentially motivating new items. The program terminates when the active agenda is empty. At that point, the complete passive items represent the chart forest.

2.1.3 Probabilistic Parsing



(a) CFG (b) Parse tree (c) Inside of NP

Figure 2: A parse tree according to a CFG, and the inside probability of the non-terminal NP.

Parsing algorithms for PCFGs (e.g. CKY and Earley) return a chart forest, which is a representation of $P(x, d)$, i.e. a joint distribution with x fixed. When reasoning about the derivations of a string x, one should, for the purpose of sampling, focus on the conditional distribution $P(d \mid x)$ rather than the joint distribution. Complying to the laws of probability theory, the conditional distribution $P(d \mid x)$ can be rewritten as:

$$P(d \mid x) = \frac{P(d, x)}{P(x)} \tag{4}$$

Furthermore, according to equation (3), $P(x)$ is equal to $\sum_{d \in D(x)} P(x, d)$. Therefore, equation (4) can be rewritten as:

$$P(d \mid x) = \frac{P(d, x)}{\sum_{d' \in D(x)} P(x, d')} \tag{5}$$

Algorithm 1 Topological sort

```
function TopSort(forest)
    $S \leftarrow$ nodes with no dependencies
    $D \leftarrow$ mapping between nodes and their direct dependencies
    $L \leftarrow$ top ordered nodes

    while $S$ is not empty do
        node = pop(S) $\leftarrow$ remove and return a node from $S$
        $L = L + node \leftarrow$ add node to the tail of $L$

        for parent in dependencies(node) do
            D(parent).remove(node) $\leftarrow$ remove node from D(parent)

            if D(parent) is empty then
                $S = S + parent \leftarrow$ add parent to $S$, the nodes with no dependencies
            end if
        end for
    end while
    return $L \leftarrow$ return the topological ordered nodes
end function
```

Consequently, in order to sample from $P(d \mid x)$, the distribution of derivations (forest) has to be normalized by the sum over all derivations within it. The quantity $P(x) = \sum_{d \in D(x)} P(x,d)$, can be efficiently computed via dynamic programming utilizing the *inside recursion*:

$$I(v) = \begin{cases} 1 & \text{, if } BS(v) = \emptyset \\ \sum_{e \in BS(v)} w(e) \times \prod_{u \in tail(e)} I(u) & \text{, otherwise} \end{cases} \tag{6}$$

Where the *inside probability* of a node v, $I(v)$, is 1 if the set of incoming edges to v (indicated as $BS(v)$) is empty, i.e. all terminal nodes have an inside probability of 1. Otherwise, the inside probability of a non-terminal node is the summation over the weights of all incoming edges (where $w(e)$ represents the weight of an edge e) which are multiplied with the inside probabilities of all the nodes within their tail[3]. More generally, the inside probability of a phrase is the probability that a certain non-terminal node spans a sequence of terminal symbols, an illustration is provided in figure 2c. The inside recursion can be efficiently computed in a bottom-up pass through the forest, assuming that the forest is acyclic[4]. Therefore, the forest determined by a parser needs to be sorted in order to utilize the inside recursion. Partial ordering algorithms exist for this

[3]A node is equivalent to a labelled input span. A complete item of the form $[A \rightarrow \alpha\bullet, i, j]$ in CKY and Earley represents an incoming edge to the node A_{i-j}. The nodes in α represent the edge's tail. This view of the forest is consistent with the notion of B-hypergraphs presented in (Gallo et al., 1993).

[4]Cycles may arise from unary rules (i.e. rules of the form $A \rightarrow B$). This present study does no deal with such grammars. For a more generally treatment robust to cycles see (Goodman, 1999).

purpose. Initially, an explanation of an ordering algorithm will be presented. Thereafter, the explanation of the implementation of the inside recursion.

Algorithm 1 illustrates topological sorting (Cormen et al., 2001), a partial ordering algorithm which runs in linear time with the size of the input forest. It sorts the given forest into a hierarchical order. The algorithm starts with a set S of nodes with no dependencies (line 2), nodes that do not appear in the left-hand side of any rule of the derivation, i.e. terminal nodes. In addition, a set D consisting of the mapping between nodes and their direct dependencies (line 3). Furthermore, it initiates with an empty list L (line 4) that will contain the sorted nodes. It continues by withdrawing nodes from S one at a time (line 7), while adding them to the set of sorted nodes (line 8). Additionally, these nodes are removed from the dependencies of their parents (line 11), i.e. the specific nodes are removed from any rule where they appear in the right hand side of the rule. A parent node can be added to S whenever its set of dependencies is depleted (line 14). This algorithm proceeds until the set of nodes with no dependencies is empty, indicating that all nodes of the forest have been sorted.

Algorithm 2 Inside algorithm

```
function INSIDE(forest, TopSort)
    for node in TopSort(forest) do
        incoming ← the set of incoming edges to node

        if not incoming then
            inside[node] = $\bar{1}$ ← terminal nodes have an inside weight of 1
        else
            inside[node] = $\bar{0}$ ← initiate nodes with an inside weight of 0

            for edge in incoming do
                p = w(edge) ← including the edge's own weight

                for child in the edge's tail do
                    p = p $\bar{*}$ inside[child] ← multiply with the child's inside weight
                end for
                inside[node] = inside[node] $\bar{+}$ p ← accumulate for each edge
            end for
        end if
    end for
    return inside ← return a dictionary mapping a node to its inside weight
end function
```

The inside recursion can be implemented via a dynamic programming algorithm, the Inside algorithm (Baker, 1979) (Lari and Young, 1990), which runs in linear time with the size of the forest; it is illustrated as algorithm 2. The algorithm processes nodes one at the time, given that the nodes are topological sorted. It initiates by retrieving all incoming edges for each node it processes (line 3), i.e. the set of rules where the current node appears in the left-hand side of the rule. Subsequently, a probability of 1 will be assigned to nodes without any incoming edges (line 6), i.e. terminal nodes. In contrast, nodes with dependencies (i.e. non-terminals) are initialized with a probability of 0 (line 8). Additionally, the weight of each incoming edge of that node is multiplied by the inside probabilities of the child nodes that appear in the right hand side of the specific edge (line 14), and accumulated by the inside probability of the current node (line 16). This algorithm terminates when all nodes have been progressed. Consequently, it returns a table containing a mapping from nodes to their inside weights (line 20).

2.2 Sampling methods

Sampling finds several applications within NLP, such as estimating expectations and approximating intractable objectives, for instance, the most probable parse problem in the context of latent PCFGs.

Two sampling techniques are presented in this section. Firstly, sampling from the inverted CDF, which is applicable when an **exact** representation of the target distribution is known (e.g. in the case of the MC sampling method proposed by Chappelier and Rajman (2000)). Secondly, a technique called slice sampling, which is a form of auxiliary variable MCMC and does not require access to the exact distribution at all times.

2.2.1 Inverted CDF

Provided that the exact PDF is known, a simple and efficient way to sample from arbitrary distributions is by sampling from the inverse CDF.

For a discrete random variable $x \in X = \{x_1, ..., x_n\}$, where the indices define an arbitrary (though fixed) order of events in X, with distribution $P_n \equiv P(n_k)$, the CDF is given by:

$$F_k = \sum_{i=1}^{k} P_i \tag{7}$$

Whenever $F_k > 0$ for all $x_k \in X$, F_k has an inverse given by:

$$F^{-1}(u) = k, \text{ where } F_{k-1} < u \leq F_k \tag{8}$$

The computation of $F^{-1}(u)$ can be realized by constructing the table $\{(k, F_k) \mid 0 \leq k \leq N\}$. In the case of sampling PCFGs, a table is constructed for every node (i.e. left-hand side of a rule along with the start and end positions) where each incoming edge represents an event n_k and F_k is given by inside weights.

To simulate the distribution of interest, one can draw a uniform random number $U \in [0, 1]$ (domain of F^-1) and look up its inverse value $F^{-1}(U)$.

2.2.2 Slice Sampling

Slice sampling, as described by Neal (2003), is based on the principle that a univariate distribution can be sampled by sampling uniformly from the 2D region under its density function. A Markov chain can be constructed that converges to this uniform distribution; this can be achieved by alternately sampling uniformly from the vertical interval from zero up to the density at the current point and sampling uniformly from the horizontal slice defined by the current vertical position.

An example of slice sampling a univaritate distribution is illustrated in figure 3 to aid the described intuition. It illustrates a distribution for a variable x, whose density function is proportional to $f(x)$. The height of $f(x)$ corresponds to the likelihood of that point. This distribution of x can be sampled by uniformly sampling points from the region that lies below $f(x)$. In addition, a slice variable u_s is introduced that is sampled uniformly from the vertical interval $[0, f(x))$, thereby defining the horizontal 'slices' of the distribution of x, i.e. a horizontal slice is defined by the region $S = \{x : u_s < f(x)\}$. Consequently, samples can be drawn uniformly from the region S, thereby reducing the complexity of sampling. This technique can be extended to a multivariate distribution by applying it to each variable in turn.

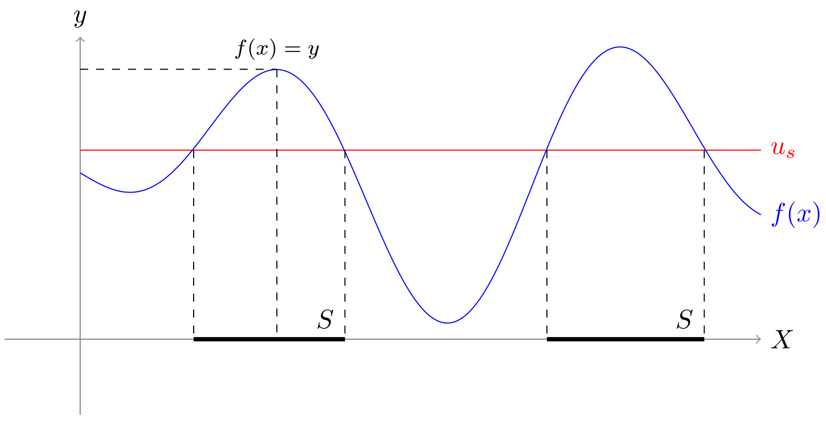


Figure 3: Illustation of Slice Sampling.

This sampling technique can be used to reduce the complexity of sampling PCFGs by limiting the scope of the chart-based parsing algorithm to high probability regions, 'surrounding' a given derivation. Consequently, samples will be drawn from a reduced space of possible parse tree derivations.

3 Research method

This chapter consists of the elaboration of the conducted research method. MC and slice sampling for PCFGs will be explained along with some design choices that were made. Furthermore, this chapter will conclude with an explanation of the evaluation methods that were used during this research.

3.1 MC Sampling for PCFGs

Chappelier and Rajman (2000) introduced a sampling technique which is applicable to PCFGs based on the principle of ancestral sampling, a form of unbiased sampling which produces independent samples from the distribution of derivations.

Chappelier and Rajman (2000) divided their approach in two phases. Firstly, an *analysis* phase, where a compact representation of all possible derivations of the input is created. Secondly, an *extraction* phase, where the results of the former representation are extracted, e.g. extracting the most probable derivation. The analysis phase can always be achieved within cubic time with respect to the input size. However, the extraction phase might lead to an NP-hard problem (Chappelier and Rajman, 2000).

The analysis phase consists of a bottom-up filling of a parse chart with items that represent all possible sub-derivations of all possible substrings of the input sentence, i.e. they utilize CKY, a dynamic parsing algorithm described in section 2.1.2 of this thesis.

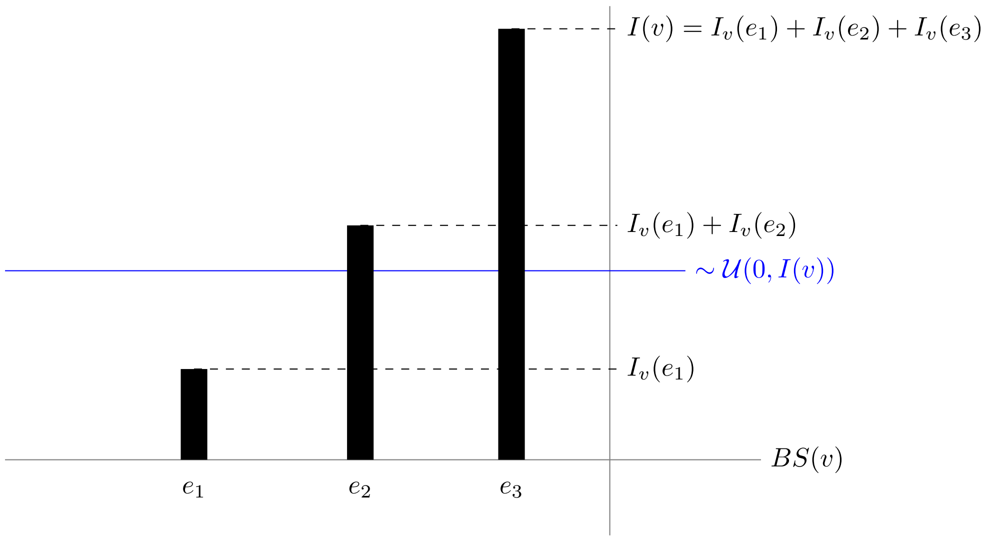
The simplest way to sample from $P(d \mid x) = \frac{P(d,x)}{\sum_{d' \in D(x)} P(x,d')}$, if the target distribution is exactly represented, is to sample from its inverted CDF, as described in section 2.2.1 of this thesis. Chappelier and Rajman (2000) obtain an exact representation of $P(x, d)$ by running CKY.

The quantity $P(x) = \sum_{d \in D(x)} P(x, d)$ can be obtained by utilizing the Inside algorithm, as described in section 2.1.3 of this thesis. More specifically, for any node v, the inside of v represents the sum of $P(x, \tau_v)$ for every subtree τ_v rooted at v. Therefore, the Inside algorithm can be used to compute the CDF with respect to each and every node.

The extraction phase consist of randomly drawing samples from the inverted CDF. A strategy called ancestral sampling is used in order to sample rewriting rules conditioned on their left-hand side symbol, taking into account the value of the CDF associated with each and every symbol in the rule, for each and every rule that can rewrite the given symbol. This is achieved by sampling from a multinomial distribution proportional to $I_v(e) = w(e) \prod_{u \in tail(e)} I(u)$, as illustrated in equation (9).

$$e \sim \frac{I_v(e)}{\sum_{e' \in BS(v)} I_v(e')} \tag{9}$$

Figure 4 illustrates how this strategy is realised. A probabilistic threshold is determined uniformly from the interval of 0 and the total inside of the current left-hand side symbol v, $I(v)$. All the inside probabilities of all incoming edges of v are accumulated each at the time, i.e. they form the CDF, until the weight of this probability intersects with the random threshold. The *image* of this graph is the inverse CDF, therefore, the intersection of the probabilistic threshold with an inside weight of an edge is in fact sampling from the inverted CDF. In the example shown below, the CDF associated with e_2 intersects with the probabilistic threshold and is thereby sampled.



$I(v) = I_v(e_1) + I_v(e_2) + I_v(e_3)$
$I_v(e_1) + I_v(e_2)$
$\sim \mathcal{U}(0, I(v))$
$I_v(e_1)$
$BS(v)$
e_1
e_2
e_3

Figure 4: Sampling from the inverted CDF.

This random sampling algorithm functions top-down and in linear time. It consist of recursively choosing a rule of the current item at random, starting at the root item until the bottom items (i.e. words) are reached and a complete derivation is obtained.

3.2 Slice Sampling for PCFGs

As already mentioned in section 2.2.2 of this thesis, the technique of *slice sampling* (Neal, 2003) can be used to reduce the complexity of sampling PCFGs by limiting the scope of the chart-based parsing algorithm to high probability regions. Consequently, samples will be drawn from a reduced space of possible parse tree derivations. Multiple steps have to be made in order to achieve this, all of these steps will be explained comprehensively, along with the implementation choices made during this study.

Initially, this present study augments the distribution of parse tree derivations with auxiliary variables u (vector) such that sampling from $P(u, d)$ is simpler than sampling from $P(d)$. This principle was put forward by Tanner and Wong (1987).

Furthermore, the fact is used that if samples can be drawn from a joint distribution $P(u, d)$, then samples can be obtained from the marginal distribution $P(d)$.

$$P(d) = \sum_u P(d, u) \tag{10}$$

Subsequently, Gibbs sampling (Geman and Geman, 1984) is utilized in order to sample from $P(u, d)$, where a joint distribution can be sampled by iteratively sampling from its conditionals:

$$u \sim P(u \mid d) \tag{11}$$

$$d \sim P(d \mid u) \tag{12}$$

Furthermore, $P(u \mid d)$ is chosen such that each auxiliary variable is independent of the others, i.e.:

$$P(u \mid d) = \prod_{u_s \in u} P(u_s \mid d) \tag{13}$$

The variable u is defined so that it contains a slice variable u_s for every cell s (each cell is defined by a left-hand sided symbol along with its start and end positions) of the chart.

$$S = \{(A, i, j) : 0 \leq i \leq j \leq |x|, A \in N\} \tag{14}$$

$$u = \{u_s \in \mathbb{R} \mid 0 < u_s < 1, s \in S\} \tag{15}$$

The distribution $P(u_s \mid d)$ can be chosen conveniently:

$$P(u_s \mid d) = \begin{cases} \frac{\mathbb{I}(u_s < \theta_{r_s})}{\theta_{r_s}} & \text{, if } r_s \in d \\ \beta(u_s; a, b) & \text{, otherwise} \end{cases} \tag{16}$$

Inspired by the technique of slice sampling (Neal, 2003), as shown in equation (16), slice variables are distributed uniformly between 0 and θ_{r_s}, where θ_{r_s} is a short for $P(r_s)$, if there is some evidence d of the value $P(r_s)$. Otherwise, the slice variables are simply distributed according to a Beta[5] distribution (where a and b control the shape of the distribution). Typically, b is chosen to be 1 and fixed, whereas a is chosen to vary towards zero, in order to let the Beta distribution have its mass towards zero, consequently, many derivations will compete when sampling this distribution[6]. $P(u_s \mid d)$ was chosen conveniently, it is therefore necessary to derive[7] what $P(d \mid u)$ looks like.

[5]The Beta distribution is defined between 0 and 1 and it is continuous, it is therefore convenient to model thresholds on probabilities (because probabilities also range from 0 to 1), its PDF is non-zero for every value in the interval, its shape can vary according to a and b including the uniform distribution.

[6]In this present study experiments were conducted with $a = 0.3$ and $b = 1$.

[7]This derivation was taken from (Blunsom and Cohn, 2010).

$P(d \mid u) \propto P(d,u) = P(d) \times P(u \mid d)$

$$\begin{aligned}
&= \left(\prod\nolimits_{r_s \in d} \theta_{r_s}\right) \times \begin{pmatrix} \prod\nolimits_{u_s : r_s \in d} \frac{\mathbb{I}(u_s < \theta_{r_s})}{\theta_{r_s}} \\ \times \prod\nolimits_{u_s : r_s \notin d} \beta(u_s; a, b) \end{pmatrix} && (17) \\
&= \prod_{u_s : r_s \in d} \mathbb{I}(u_s < \theta_{r_s}) \prod_{u_s : r_s \notin d} \beta(u_s; a, b) && (18) \\
&= \prod_{u_s : r_s \in d} \frac{\mathbb{I}(u_s < \theta_{r_s})}{\beta(u_s; a, b)} \prod_{u_s} \beta(u_s; a, b) && (19) \\
&\propto \prod_{u_s : r_s \in d} \frac{\mathbb{I}(u_s < \theta_{r_s})}{\beta(u_s; a, b)} && (20)
\end{aligned}$$

The θ_{r_s} terms are cancelled in step (18). In step (19) the term $\beta(u_s; a, b)$ for $u_s : r_s \in d$ is introduced to the numerator and denominator. $\prod_{u_s} \beta(u_s; a, b)$ is removed in the final step (20) because it does not depend on d, which makes (20) proportional to (19).

These slice variables function as random thresholds on the probabilities of the rules considered in each cell s. Every rule r headed by s and whose probability θ_{r_s} is larger than the slice variable u_s is accepted, each rule that does not satisfies this condition is pruned from the dynamic program.

3.2.1 Implementation Details

$$\begin{array}{lll}
\text{Goal item:} & [\mathrm{S},0,\mathrm{n}] & \\[1em]
\text{Axioms:} & \dfrac{}{[A \xrightarrow{\theta} \bullet\alpha, i, i]} & \begin{array}{c} A \xrightarrow{\theta} \alpha \in G \\ 0 \le i \le n \end{array} \\[2em]
\text{Scan 1:} & \dfrac{[A \xrightarrow{\theta} \alpha \bullet a\beta, i, j]}{[A \xrightarrow{\theta} \alpha a \bullet \beta, i, j+1]} & \beta \neq \varepsilon \\[2em]
\text{Scan 2:} & \dfrac{[A \xrightarrow{\theta} \alpha \bullet a, i, j]}{[A \xrightarrow{\theta} \alpha a \bullet, i, j+1]} & \theta > u_{[A,i,j+1]} \\[2em]
\text{Complete 1:} & \dfrac{[A \xrightarrow{\theta} \alpha \bullet B_{j-k}\beta, i, j], [B \xrightarrow{\theta'} \gamma\bullet, j, k]}{[A \xrightarrow{\theta} \alpha B_{j-k} \bullet \beta, i, k]} & \beta \neq \varepsilon \\[2em]
\text{Complete 2:} & \dfrac{[A \xrightarrow{\theta} \alpha \bullet B_{j-k}, i, j], [B \xrightarrow{\theta'} \gamma\bullet, j, k]}{[A \xrightarrow{\theta} \alpha B_{j-k}\bullet, i, k]} & \theta > u_{[A,i,j]}
\end{array}$$

Figure 5: A modification of the Nederhof et al. parsing algorithm for slice sampling.

To restate, rather than sampling from the exact distribution of parse tree derivations $P(d)$, samples are drawn from the distribution of $P(u, d)$, where u is a vector of slice variables, which function as cutoffs on the probabilities of the rules considered in each cell s. Sampling from $P(u, d)$ can be achieved by iteratively sampling from its conditionals: $u \sim P(u \mid d)$ and $d \sim P(d \mid u)$.

The vector of auxiliary variables u is obtained during the process of parsing, where every slice variable u_s is assigned a uniform or a Beta probability, depending on the existence of the current cell s in the derivation d where it is conditioned on (i.e. the last sampled derivation). The parsing algorithms, described in section 2.1.2 of this thesis, (e.g. Earley and Nederhof et al) had to be modified for this purpose. Figure 5 illustrates a modified version of the generalized CKY (i.e. the CKY based parsing algorithm by Nederhof and Satta (2008)), where completed items are treated differently. A complete item can only be inferred if the probability of its underlying rule θ is larger than the threshold determined by u_s, as illustrated in equation (16). This modification applies to the scan 2 and complete 2 operations, as shown in figure 5.

Furthermore, derivations are sampled from the distribution $P(d \mid u)$. Analog to the MC sampling method described in the previous section, the inside probabilities had to be computed. The inside probabilities are, however, slightly different. Instead of taking the weight of an edge, the expression shown in (20) is used, i.e. $\frac{1}{\beta(u_s;a,b)}$[8]. After this modification the inside recursion remains the same. Thereafter, sampling from the derivation distribution can be achieved analog to the described sampling method in the previous section. The succeeding derivation will be conditioned on a previously sampled derivation.

Another implementation detail which is worth mentioning is the initial conditions; if every derivation is conditioned on a previously sampled derivation, then, what is the first derivation based on? Initially, the first derivation would purely be conditioned on slice variables with values from the Beta distribution, however, this results in a very slow first iteration. A solution, which is implemented, is to create initial conditions prior to deriving the first derivation. This is accomplished by reducing the size of the PCFG, and sampling a derivation based on the reduced PCFG. This derivation serves as the initial conditions, where the first derivation will be conditioned on.

3.3 Evaluation Methods

This present study focuses on very large grammars, e.g. grammars that are automatically learned from data. Specifically, the grammars used in this research are for the purpose of an application in machine translation, namely, *reordering models.*

The structure of words within a sentence differ a lot among languages, e.g. a Dutch sentence translated to English:

[8]Observe that this defines a uniform distribution for each cell.

Dutch: ‘Ik$_1$ wil$_2$ naar$_3$ bed$_4$ **gaan$_5$**’

English: ‘I$_1$ want$_2$ to$_2$ **go$_5$** to$_3$ bed$_4$’

Where the verb ‘gaan’ is located at the end of the sentence, whereas the translational equivalent ‘go’ is located in the middle of the sentence. Machine translation would be a lot easier if the languages were *monotone*, i.e. if they had the same word order. This can be accomplished with reordering models, where the Dutch example sentence would be reordered to ‘Ik wil gaan naar bed’.

Stanojevic and Sima’an (2015) introduce a reordering model based on PCFGs whose non-terminal symbols represent permutations of their children, e.g.:

$$P123 \rightarrow x\ y\ z \Rightarrow x\ y\ z$$

$$P231 \rightarrow x\ y\ z \Rightarrow y\ z\ x$$

Where the order of the children of the first illustrated rule remains the same, but the order of the second example is rearranged according to the permutation 231.

In this present study three grammars were used, varying in size, as shown in table 1:

Grammar	Number of rules
1	$0.54{\times}10^6$
2	$1.43{\times}10^6$
3	$2.71{\times}10^6$

Table 1: Grammars varying in size.

Furthermore, a data set of input sentences is used, divided into six batches, varying in sentence length, as shown in table 2:

Batch	Sentence length (mean)	Size
1	1-10 (8)	36
2	11-20 (17)	50
3	21-30 (25)	50
4	31-40 (35)	50
5	41-50 (44)	50
6	50+ (51)	13

Table 2: Input sentences divided into six batches, varying in sentence length.

The experiments consist of combinations of the different batches and grammars for both the ancestral and slice sampling approaches. The results of these experiments will be evaluated in terms of *complexity* and *accuracy.*

The complexity will be empirically assessed in terms of sampling time with respect to the size of the grammar and the size of the input sentences. Ancestral sampling requires a complete run of CKY which runs in $\mathcal{O}(G^3n^3)$. It is therefore expected that the sampling time of ancestral sampling will increase cubically with respect to the size of the grammar and input sentence. However, the complexity of slice sampling for PCFGs is less clear, and interesting to investigate.

The accuracy will be evaluated according to two metrics. Firstly, the TAU metric, initially introduced by Kendall (1938) and later extended for reordering models in machine translation by Isozaki et al. (2010). It is a metric of rank correlation, where in the case of word order, the correlation of the word indices are compared. The returned value of TAU varies between -1 and 1, where 1 indicates a perfect correlation, -1 a perfect inverted correlation and 0 no correlation at all. Secondly, the $BLUE$ (Papineni et al., 2002) metric will be used, which computes the overlap of two permutations, by measuring n-gram matching precision (by default $BLUE$ considers n-grams for n from 1 to 4). The value $BLUE$ returns varies from 0 to 1, where 0 indicates that there was no overlap at all and 1 indicates a perfect overlap, i.e. sentences with a $BLUE$ score of 1 are similar.

4 Results

The results of the performed experiments will be presented in this chapter. Additionally, the results will be evaluated according to the evaluation criteria described in section 3.3 of this thesis, i.e. in terms of *complexity* and *accuracy*. Furthermore, during the conducted experiments it turned out that the Early parsing algorithm performed rather poorly, therefore, all experiments were achieved utilizing the generalised CKY (Nederhof and Satta, 2008) parsing algorithm.

4.1 Complexity

Figure 6 illustrates a log-log plot of slice sampling compared to ancestral sampling. The horizontal axis represents the average length of the sentences used according to the first three batches (hence the three data-points per sampler). The vertical axis represents the average time that was necessary per sample, i.e. the total sample time of a batch divided by the number of sentences and the number of samples taken. The axes are plotted in a logarithmic scale in order to ease the analyze of the complexity of both approaches, i.e. the slope of the function indicates the complexity. This experiment consisted of the first three batches (i.e. sentences up to length 30) and the first grammar mentioned in table 1. Furthermore, 100 samples were drawn from the distribution of derivations.

The first obvious observation is the average sampling time per sample, in general. Initially, slice sampling is significantly slower than the ancestral sampling approach. However, the slope of the ancestral sampling approach is significantly steeper, indicating that the complexity of the slice sampling approach is reduced with respect to the ancestral approach. One can easily observe that the ancestral approach will intersect with the slice sampling approach. Consequently, the sampling time of ancestral sampling per sample of this experiment will increase dramatically when the sentence length increases and will be beyond the sampling time of slice sampling.

The parse time (per call) of the slice and ancestral approaches is compared in figure 7,

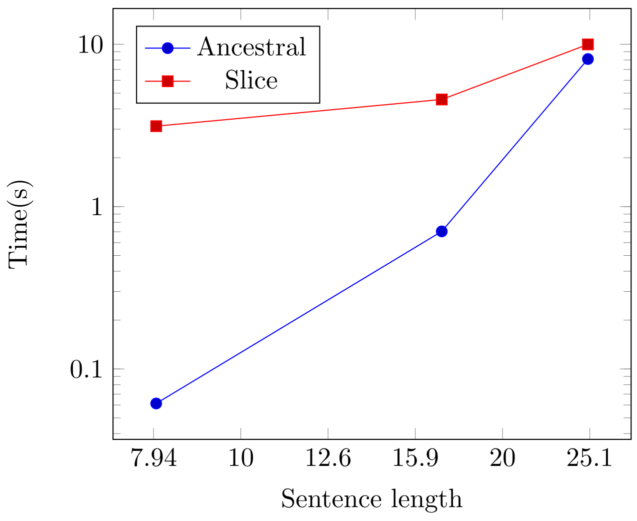


Figure 6: Slice sampling compared to ancestral sampling, where the average time per sample is plotted with respect to the mean length of the input sentences in a batch (batch 1-3). The axes are shown in logarithmic scale. This experiment is based on 100 samples and the first grammar.

with respect to the length of the input sentence. The same grammar and input sentences were used as in figure 6. Initially, the parsing time of short sentences are for both the approaches reasonable, since the number of possible derivations that can be parsed is limited for those sentences. Although, even for short sentences, the parse time of the ancestral approach is twice as high as the parse time of slice sampling. The parse time of ancestral sampling increases dramatically with respect to the length of the sentence, whereas, it is once more noticeable that the slope of slice sampling is less steep than the slope of the ancestral sampling approach indicating that the complexity of slice sampling is reduced with respect to ancestral sampling. However, although the parsing time of slice sampling is significant lower than the parsing time of ancestral sampling, slice sampling requires k parses, where k is the number of samples taken, in contrary to ancestral sampling which always requires one parse.

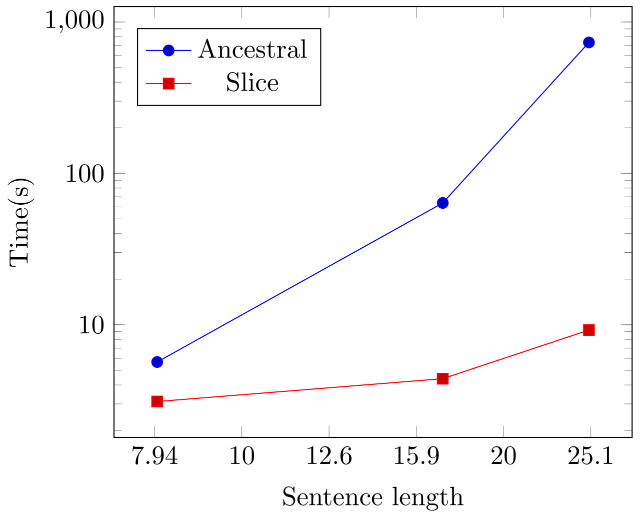


Figure 7: Slice sampling compared to ancestral sampling, where the parsing time per call is compared, with respect to the mean length of the input sentences in a batch (batch 1-3). The axes are shown in logarithmic scale. This experiment is based on 100 samples and the first grammar.

4.2 Accuracy

Data set		Ancestral Sampling		Slice Sampling	
Batch	Grammar Size	BLUE	TAU	BLUE	TAU
1	$0.54{\times}10^6$	0.61	0.79	0.50	0.86
2	$0.54{\times}10^6$	0.44	0.69	0.43	0.77
3	$0.54{\times}10^6$	0.40	0.63	0.41	0.73
4	$0.54{\times}10^6$	-	-	0.40	0.72
5	$0.54{\times}10^6$	-	-	0.41	0.71
6	$0.54{\times}10^6$	-	-	0.44	0.72

Table 3: BLUE and TAU scores of ancestral and slice sampling, after 100 samples for grammar 1.

Table 3 and 4 consists of the *BLUE* (Papineni et al., 2002) and *TAU* (Isozaki et al., 2010) scores of the ancestral and slice sampling approaches. The conducted experiments in table 3 were based on all batches of the input sentences and the first grammar, both the batches and grammars are specified in table 2 and 1 respectively. It is noticeable that the last three batches of ancestral sampling are omitted, this is due to the fact that those batches were impracticable for this approach[9]. In terms of the *BLUE* score, the ancestral sampling approach scores slightly

[9]A remote server was utilized for the conducted experiments. This machine has over 200 gigabyte of memory and 20 cores, and it ran out of memory after the third batch while running ancestral sampling. Whereas the slice sampling method only consumes approximately 3 gigabyte of memory.

better than the slice sampling approach, however, the loss is not significantly. In terms of TAU, the slice sampler seems to score slightly higher than the ancestral sampling approach.

Most likely the differences observed between the two samplers, particularly, the seemly superior performance of the slice sampler, is not statistically significant, which means that they are most likely equivalent (their confidence intervals overlap). In principle, if the confidence intervals are not computed, there is no way of knowing for certain whether the variance of the slice sampler is close to the variance of the exact sampler. That is, in principle there is a chance that the confidence intervals for the slice sampler are much larger than those of the exact sampler, which would imply that the variance of the slice sampler is undesirably large. However, if that was the case, there most likely would have been a large variance across batches, i.e. for some batches slice sampling would be much better than ancestral sampling, for others much worse. As the conducted research consisted of many batches that were independently tested, and there were no random ups and downs observed, the most likely explanation is that the slice sampler's variance is acceptable. This would imply that the slice sampler is indeed performing very closely to the exact one for these data sets.

Data set		Ancestral Sampling		Slice Sampling	
Batch	Grammar Size	BLUE	TAU	BLUE	TAU
1	1.43×10^6	0.64	0.83	0.62	0.87
2	1.43×10^6	0.48	0.75	0.43	0.84
3	1.43×10^6	0.44	0.71	0.40	0.79
1	2.71×10^6	0.67	0.86	0.57	0.94
2	2.71×10^6	0.43	0.78	0.38	0.85

Table 4: BLUE and TAU scores of ancestral and slice sampling, after 100 samples, for grammar 2 and 3.

Table 4 consists of some additional experiments with the remaining two grammars. Although not all batches have been included in these experiments, it is noticeable, with respect to the results in table 3, that the trend of the performance of the slice sampling approach remains the same among different sizes of grammars.

5 Conclusion

Determining the most probable parse of a sentence is a common task within NLP. However, this maximization task corresponds to an NP-hard problem (Sima'an, 1996) in the particular case of latent PCFGs. Sampling methods offer a tractable and sound alternative in order to deal with this problem, where a reduced though representative statistical sample of derivations is used to reason about underlying parse trees.

MC sampling approaches, such as ancestral sampling, often require the exact representation of the target PDF. However, the computation of this exact representation requires a complete run of a chart-based parsing algorithm (e.g. CKY), which has a complexity of $\mathcal{O}(G^3 n^3)$. Consequently, MC sampling approaches will be impracticable when grammars or sentences become too large, which is often the case with grammars that are automatically learned from data (e.g. latent PCFGs).

In this present study it is argued that a complete run of a chart-based parsing algorithm is wasteful, in the sense that most of the items enumerated by these algorithms are unlikely to be sampled.

An MCMC sampling approach, specifically slice sampling, is proposed as an alternative to regular MC sampling approaches. Slice sampling reduces the dimension of the problem by 'slicing' the distribution over derivations into high probable regions; samples could then be drawn from those high probable regions rather than from the whole distribution of derivations.

During this research both the MC and the slice sampling approaches are implemented and compared with one another. Thereby investigating two questions. Firstly, to what extend can slice sampling reduce the complexity of sampling PCFGs with respect to MC sampling? Secondly, to what extend will slice sampling preserve the accuracy of MC sampling?

The results of this present study were particularly encouraging. The sampling and parse time of the slice and MC sampling approaches were plotted (in a log-log plot) and compared with each other. An empirically analysis showed that the slope of the slice sampling approach was less steep than the slope of the utilized MC sampling approach (i.e. ancestral sampling). Therefore, it can be concluded that slice sampling reduced the complexity of sampling PCFGs in comparison to the utilized MC sampling approach. However, the overall sampling time of slice sampling was rather slow, so there is still a lot to gain. Subsequently, the accuracies of both sampling approaches are compared in terms of $BLUE$ (Papineni et al., 2002) and TAU (Isozaki et al., 2010) scores. The accuracy of slice sampling was for both the $BLUE$ and TAU scores very close to the accuracy of the MC sampling approach, even though the experiments were based on just 100 samples. So in order to answer the second research question, slice sampling seems to preserve the accuracy of the MC sampling approach, since the difference in accuracy was not significantly.

Although the results are particularly encouraging, there is still enough room for future work. Firstly, it is necessary to optimize the code since the slice sampler is still rather slow. This should be done by improving the utilized data-structures as well as using static typing, e.g. C^{++} or Cython rather than Python. Secondly, by conditioning on sets of derivations rather than on one single derivation, this would enable the possibility to draw multiple samples per iteration. Finally, the sampler could be parallelized, this can be accomplished in multiple ways. For instance, by means of a heuristic version, where multiple seed derivations would sample parallel to each other. Another approach for parallelizing the sampler is by integrating the MCMC approach with other MC techniques, which are parallel in nature, such as importance sampling and rejection sampling. Finally, the confidence intervals should be computed in order to confirm that the slice sampler is indeed performing very closely to the exact one.

References

Baker, J. K. (1979). Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550, Boston, MA.

Blunsom, P. and Cohn, T. (2010). Inducing synchronous grammars with slice sampling. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 238–241. Association for Computational Linguistics.

Chappelier, J.-C. and Rajman, M. (2000). Monte-carlo sampling for np-hard maximization problems in the framework of weighted parsing. In *Natural Language ProcessingNLP 2000*, pages 106–117. Springer.

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

Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Gallo, G., Longo, G., Pallottino, S., and Nguyen, S. (1993). Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(6):721–741.

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

Hopcroft, J. E. and Ullman, J. D. (1969). *Formal Languages and Their Relation to Automata*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Isozaki, H., Hirao, T., Duh, K., Sudoh, K., and Tsukada, H. (2010). Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 944–952, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kallmeyer, L. (2010). *Parsing Beyond Context-Free Grammars*. Cognitive Technologies. Springer Heidelberg.

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, 30(1/2):81–93.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside–outside algorithm. *Computer Speech and Language*, 4:35–36.

Neal, R. M. (2003). Slice sampling. *Annals of statistics*, pages 705–741.

Nederhof, M.-J. and Satta, G. (2008). Probabilistic parsing. In *New Developments in Formal Languages and Applications*, pages 229–258. Springer.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006a). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia. Association for Computational Linguistics.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006b). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Robert, C. P. and Casella, G. (2004). *Monte Carlo Statistical Methods.* Springer Texts in Statistics. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Shieber, S. M., Schabes, Y., and Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.

Sima'an, K. (1996). Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 1175–1180. Association for Computational Linguistics.

Sima'an, K. (2003). On maximizing metrics for syntactic disambiguation. In *Proceedings of the International Workshop on Parsing Technologies*, IWPT'03.

Stanojevic, M. and Sima'an, K. (2015). Reordering context-free grammar induction. To appear.

Tanner, M. A. and Wong, W. H. (1987). The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82(398):528–540.