Probabilistic (k,l)-Context-Sensitive Grammar Inference with Gibbs Sampling Applied to Chord Sequences

Henrique Barros Lopes[a] and Alan Freitas[b]
Departamento de Computação, Universidade Federal de Ouro Preto, Campus Universitário Morro do Cruzeiro,

Keywords: Grammatical Inference, Computer Music, Chord Sequence Learning, Probabilistic Context-Sensitive Grammars, Machine Learning, Algorithmic Composition, Artificial Intelligence.

Abstract: Grammatical inference in computer music provides us with valuable models for fields such as algorithmic composition, style modeling, and music theory analysis. Grammars with higher accuracy can lead to models that improve the performance of various tasks in these fields. Recent studies show that Hidden Markov Models can outperform Markov Models in terms of accuracy, but there are no significant differences between Hidden Markov Models and Probabilistic Context-Free Grammars (PCFGs). This paper applies a Gibbs Sampling algorithm to infer Probabilistic (k,l)-Context-Sensitive Grammars (P(k,l)CSGs) and presents an application of P(k,l)CSGs to model the generation of chord sequences. Our results show Gibbs Sampling and P(k,l)CSGs can improve on PCFGs and the Metropolis-Hastings algorithm with perplexity values that are 48% lower on average (p-value 0.0026).

1 INTRODUCTION

The inference of probabilistic grammars is a machine learning approach applied in a myriad of fields. Works inferring grammars are used, for example, to validate protocols (Nouri et al., 2014; Mao et al., 2012) and to infer flow diagrams (Breuker et al., 2016). Computational linguistics is an important field that makes extensive use of grammatical inferences (Heinz and Rogers, 2010; Jarosz, 2013; Jarosz, 2015a; Jarosz, 2015b). This field uses inferred grammars to learn patterns and relationships between letters in a word or words in a phrase. These patterns depend on the expressiveness of the inferred grammar. For instance, context-sensitive grammars may define relationships between words in English spoken language (Jäger and Rogers, 2012).

While grammar defines rules that help people write and speak clearly and correctly, music has hierarchical theories, e.g., functional harmony, that guide composers to write songs pleasant to listeners. This similarity led researchers from linguistics and music to use similar strategies to guide their work (Forsyth and Bello, 2013).

In computer music, the grammatical inference is useful to fields such as algorithmic composition, style modeling, and musical theory analysis. In Section 2, we describe many applications of grammatical inferences in computer music along with the grammatical classes inferred in each work. We observed that Bayesian inference works better than other approaches to infer grammars, and the use of Probabilistic (k,l)-Context-Sensitive Grammars (P(k,l)CSG)s has the potential to improve the state of the art in chord sequence modeling.

The Bayesian algorithms Gibbs Sampling (GS) and Metropolis-Hastings (MH) (Johnson et al., 2007; Tsushima et al., 2018a) have been used to infer Probabilistic Context-Free Grammars (PCFG), which are in the second level of Chomsky's hierarchy. On the other hand, P(k,l)CSGs have been applied to model part-of-speech tags, as in (Shibata, 2014), who has proposed an MH algorithm to infer both PCFGs and P(k,l)CSGs. The experiments showed a P(k,l)CSG with $(k,l) = (1,0)$ achieved smaller perplexity values than the actual state of the art algorithms.

In Section 3, this paper proposes a blocked GS algorithm to infer P(k,l)CSGs to model chord sequences. We show the details of the GS algorithm for PCFGs and describe how it can be modified to infer a P(k,l)CSG.

[a] https://orcid.org/0000-0002-2185-1635
[b] https://orcid.org/0000-0002-1266-0204

Lopes, H. and Freitas, A.
Probabilistic (k,l)-Context-Sensitive Grammar Inference with Gibbs Sampling Applied to Chord Sequences.
DOI: 10.5220/0010195905720579
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 572-579
ISBN: 978-989-758-484-8

In Section 4, we describe how we conducted our experiments by inferring PCFGs and P(k,l)CSGs to model the Billboard data set (Burgoyne et al., 2011) using the GS and MH algorithms. In Section 5, we conclude our work and present suggestions for future work.

2 GRAMMATICAL INFERENCE

2.1 Sub-regular Grammars

Markov Models (MMs) are probabilistic models describing a sequence of possible events in which each event's probability depends only on the state attained in the last o events (Gagniuc, 2017). When dealing with grammar, these events are symbols of a word, where a word is a sequence of grammar terminal symbols. Equation 1 defines a MM of order o, where X_{n+1} denotes the probability of an event. These models are equivalent to n-grams.

$$P(X_{n+1}|X_n,\ldots,X_0) = P(X_{n+1}|X_n,\ldots,X_{n-o}) \quad (1)$$

Although computer music researchers often use MMs (Bhattacharjee and Srinivasan, 2011; Nakamura et al., 2018; Takano and Osana, 2012; Trochidis et al., 2016), it is possible to represent the test data set with more expressive models. The perplexity measure ρ, defined in Equation 2, informs the joint probability of a grammar yielding each of the word w in the test set W.

$$\rho = e^{\left(\left(-\frac{1}{|W|}ln(\prod_{w\in W}P(w|\theta)\right)\right)} \quad (2)$$

2.2 Finite Automata

A Probabilistic Finite Automaton (PFA) is a tuple $(Q,\sum,\sigma,q_0,\theta)$ where (i) Q is a finite set of states; (ii) $\sum$ is the grammar alphabet; (iii) $(\sigma : QX\sum \to Q)$ is a transition function; (iv) q_0 is its initial state; and (v) θ is a probability vector such that $|\theta|$ is the number of transitions in the grammar.

Extensions of Hidden Markov Models (HMM) and PFA, such as Mondrian Hidden Markov Models (MHMMs) (Nakano et al., 2014), and variable Markov oracles (VMO-HMM) (Wang and Dubnov, 2017), can achieve smaller perplexity values than MMs (Tsushima et al., 2018a). These models have been used for chord sequences (Tsushima et al., 2018a; Wang and Dubnov, 2017), drums pattern classification (Blaß, 2013), and musical signal processing (Nakano et al., 2014).

2.3 Context-Free Grammars

A Probabilistic Context-Free Grammar (PCFG) is a 5-tuple (N,T,R,S,θ) where (i) N is the set of non-terminals; (ii) T is the set of terminals; (iii) R is the set of transition rules; (iv) $S \in N$ is the initial symbol; and (v) θ is a probability vector such that $|\theta|$ is the number of transitions. Each rule takes the form $A \to \alpha$, where $A \in N$, and $\alpha = [\alpha_1,\ldots] : \alpha_i \in N \cup T$.

PCFGs have been used for chord sequences (Tsushima et al., 2018a), improvising follow-ups (Kitani and Koike, 2010), and harmonization (Tsushima et al., 2017). These grammars have been inferred with Gibbs Sampling (GS), SEQUITUR (Kitani and Koike, 2010), and the Expectation-Maximization (EM) algorithm (Johnson et al., 2007; Tsushima et al., 2018a; Tsushima et al., 2017; Tsushima et al., 2018b). Bayesian approaches like GS have been shown to achieve smaller values of perplexity than grammars inferred with the EM algorithm (Geman and Geman, 1984).

2.4 (k,l)-Context-Sensitive Grammars

A Probabilistic Context-Sensitive Grammar (PCSG) is a 5-tuple (N,T,R,S,θ) where (i) N is a the set of non-terminals; (ii) $T : T \cap N = \varnothing$ is the set of terminals; (iii) R is the set of rules; (iv) $S \in N$ is the initial symbol; and (v) $\theta : |\theta| = |R|$ is a probability vector. Each rule takes the form $\gamma \to \alpha$, where $\alpha = [\alpha_1,\ldots] : \forall i \alpha_i \in N \cup T$ and $\gamma = [\gamma_1,\ldots] : \exists i \gamma_i \in N$. In other words, a PCSG considers which symbols are surrounding a non-terminal to decide which rule should be used.

To avoid the problem of sequences of infinite length in PCSGs, we can use Probabilistic (k,l)-Context-Sensitive Grammars (P(k,l)CSGs) (Shibata, 2014). A P(k,l)CSG is a PCSG where the rules follow the form $\gamma_l A \gamma_r \to \gamma_l \alpha \gamma_r$ such that $A \in N, \gamma_l \in N \cup T, \gamma_r \in N \cup T, \alpha \in N \cup T$. The variables γ_l and γ_r define a rule's right and left contexts, respectively, and the variables k,l indicate the maximum length of these contexts. Thus, a P$(0,0)$CSG is equivalent to a PCFG.

P(k,l)CSGs have been inferred with the Metropolis-Hastings (MH) algorithm (Shibata, 2014). The GS algorithm could be applied to these grammars despite its low performance. On the other hand, GS has the potential to use all the data in the training data set to estimate probabilities.

3 INFERRING PCFGs AND PCSGs WITH Gibbs SAMPLING

This paper proposes the use of a GS algorithm to infer P(k,l)CSGs. The GS algorithm can use all the training data so that we might achieve lower perplexity values for our grammar (see Equation 2). The source code for this work is available in our repository[1].

In Section 3.1, we briefly discuss the GS algorithm. Section 3.2 discusses the general GS algorithm proposed by (Johnson et al., 2007) for inferring PCFGs. In Section 3.3, we use this algorithm as a reference for a GS algorithm for inferring P(k,l)CSGs.

3.1 Gibbs Sampling

GS (Geman and Geman, 1984) is a Monte Carlo Markov Chain (MCMC) method that allows us to obtain samples from sets with multivariate probability distributions. Algorithm 1 describes the method. Suppose we want k samples from a multivariate random variable $\mathbf{X}$. For each of these k samples (Line 1), we generate some initial value $\mathbf{X}^i$ (Line 2) and iteratively update this value with new samples (Line 3). The initial values $\mathbf{X}^i$ can be determined randomly or by some other algorithm.

In Line 4, we determine the current probability distribution for the vector component $x^i_j \in \mathbf{X}^i$. Because we have not sampled values for x^i_k such that $k > j$ yet, note the conditional probability p considers $\mathbf{X}^i$ for the indices $k < j$ and $\mathbf{X}^{i-1}$ for $k > j$. In Line 5, we sample the component x^i_j from this distribution d.

Algorithm 1: Gibbs Sampling.

```
   Data: k
   Result: {X^1, ..., X^k}
1  for i=1 to k do
2  |  X^i ← initial();
3  |  for j=1 to n = |X| do
4  |  |  d ← p(x^i_j | x^i_1, ..., x^i_{j-1}, x^{i-1}_{j+1}, ..., x^{i-1}_n);
5  |  |  x^{(i)}_j ← sample(d);
6  |  end
7  end
```

The sample sequence $\mathbf{X}^1, \mathbf{X}^1, ..., \mathbf{X}^k$ forms an ergodic Markov Chain, where its stationary state represents the target distribution p.

[1] https://github.com/henriqueblopes/grammatical-inferences

3.2 Gibbs Sampling for PCFGs

Given the training data, the GS inference of PCFGs (Johnson et al., 2007) defines a target distribution over a set of syntactic trees. The GS algorithm runs until the target distribution p is known, and θ represents the training set appropriately.

Note that the sequence α of a PCFG can be any combination of symbols from the set $N \cup T$, making R infinite. However, we need to estimate $\theta_r, \forall r \in R*$ to perform the Bayesian inference of a PCFG. That is, we need R to be finite.

Theorem 1 (Chomsky's Normal Form Theorem)**.** *Any context-free language can be yielded by a grammar whose rules are of the form $A \rightarrow BC$ or $A \rightarrow a$ such that $A, B, C \in N$ and $a \in T$.*

With Theorem 1, we ensure that if N and T are finite, then R is finite. Thus, θ is finite. Each word a grammar yields has at least one syntactic tree. An applied rule sequence from the grammar's starting symbol until the complete word yielding defines this tree. We can use Bayes' rule to calculate the syntactic tree's t_i probability, given a word w_i as in Equation 3 (Johnson et al., 2007).

$$P(t_i|\theta, w_i) = \frac{P(w_i|\theta, t_i)P(t_i|\theta)}{P(w_i|\theta)} \tag{3}$$

Let W be the set of all words in the training data set. If we calculate the product in Equation 4 for $P(t_i|\theta, w_i)$ and all $w_i \in W$, we will have the joint probability $P(T|\theta, W) = P(t_1, t_2, ...t_n|\theta, W)$, which is the joint probability of all trees in the set $T = \{t_1, t_2, ...t_n\}$ that yields the set W. The GS algorithm can use the distribution $P(T|\theta, W)$ as a target distribution.

$$P(T|\theta, W) = \prod P(t_i|w_i, \theta) \tag{4}$$

As each tree probability depends on the rules' probabilities, finding the distribution $P(T|\theta, W)$ is the same as finding the best θ that represents W. With the target distribution $P(T|\theta, W)$, we need to achieve the following goals for a PCFG with GS:

- Calculate the prior probabilities θ;
- Sample from $P(T|\theta, W)$;
- Calculate the posterior probabilities of θ.

We take the prior probabilities as a Dirichlet distribution (Johnson et al., 2007) because it is conjugate to the PCFG's tree distribution. Which means that if a prior distribution over trees θ is a Dirichlet distribution, then the posterior distribution is a Dirichlet distribution (Johnson et al., 2007). This property facilitates the posteriors' calculations.

The *inside algorithm* calculates the inside table (Johnson et al., 2007). This table represents the probabilities of a given non-terminal yielding each subword of w_i. A sample of $P(T|\theta,W)$ is then obtained by sampling a tree $t \in T_{w_i}$ yielding w_i for each i, which can be done with Johnson's Sample function (Johnson et al., 2007). This function makes use of the inside table to randomly draw a triplet (j,B,C), where j is the division point of w_i, B represents a rule for sampling t, and C represents another rule for sampling t.

With the inside table and the sample function, we can perform the GS for our PCFG. The parameters are the prior θ, a PCFG, the maximum number of iterations, and a training data set TD. We first initialize a tree set. Then, for each $w_i \in W$, we calculate the inside table and sample a tree. We update the probabilities by calculating the posterior distribution using the counter of each rule used to sample trees and repeat these steps according to the maximum number of iterations. At this point, the algorithm only updates the vector θ after sampling all trees. This variation is known as blocked Gibbs Sampling.

3.3 Gibbs Sampling for P(k,l)CSGs

Let w denote a word yielded by a PCFG rule sequence $r_1, r_2, ..., r_n$. Then the tree probability is the product of all rule probabilities. In a P(k,l)CSG, we define the context-sensitive probabilities as in Equation 5 (Shibata, 2014).

$$P(\gamma_l A \gamma_r \rightarrow \gamma_l \alpha \gamma_r) = P(A \rightarrow \alpha | \gamma_l, \gamma_r) \quad (5)$$

Equation 5 allows a PCFG to be transformed into a P(k,l)CSG by calculating the probabilities of each rule marginalized by the contexts γ_l and γ_r. This transformation is important because it allows us to use Theorem 1 to turn the rule set finite. Therefore, we can generate a P(k,l)CSG with all possible rules given by T and N.

The sampling of $P(T|\theta,W)$ in a P(k,l)CSG is similar to the sampling in PCFGs. The main difference is that when we use a rule of the form $\gamma_l A \gamma_r \rightarrow \gamma_l BC\gamma_r$, we must take into account the right context of B given by the sequence $C\gamma_{d-1}$, where γ_{d-1} is the context γ_r without its last symbol. Another difference is that we must know whether a rule with the left or right context is valid. For instance, at the beginning of the algorithm, the starting symbol S does not have a context. So we must forbid rules of the form $\gamma_l S\gamma_r \rightarrow \gamma_l BC\gamma_r$ at this step.

Algorithm 2 (Shibata, 2014; Johnson et al., 2007) shows the steps to do the sampling. The algorithm requires an initial word w, the left-hand side $\gamma_l A\gamma_r$ of the initial rule, a start position I for yielding rules, an final position K for yielding rules, the inside table IT, a maximum left context length k and a maximum right context length l.

Algorithm 2: Sample Tree-KL.

```
   Data: w, γ_l A γ_r, I, K, IT, k, l
   Result: R
1  begin
2      if K-I=1 then
3          R ← R ∪ {γ_l A γ_r → γ_l w_k γ_r};
4      else
5          cd ← categorical_distribution(IT);
6          (J,B,C) ← sample(cd);
7          (γ_l,γ_r) ← draw_context(k,l);
8          AP ← update_yield();
9          (γ_l,γ_r) ← adjust_context(AP);
10         R ← R ∪ {γ_l A γ_l → γ_l BC γ_l};
11         R ← sample_tree_KL(w, γ_l BC γ_{r-1}, I,
             I+J, IT, k, l);
12         AP ← update_yield();
13         (γ_l,γ_r) ← adjust_context(AP);
14         R ← sample_tree_KL(w, γ_l C γ_r,
             I+J+1, K, IT, k, l);
15     end
16 end
```

In Line 2, Algorithm 2 verifies if the starting and ending positions consist of a terminal yield. If the positions represent a terminal yield, R includes the rule $\gamma_l A\gamma_r \rightarrow \gamma_l w_k \gamma_r$, where w_k is the terminal to be yielded (Line 3). If the positions do not represent a terminal yield, Line 5 calculates a categorical distribution from IT. Line 6 samples two non-terminals (B,C) and a position J that the recursive calls to the function will use to calculate the new starting and ending positions.

Next, we need to decide which right and left contexts the yield will use. Line 7 randomly draws one left and one right context from all possible contexts according to its maximum lengths, k, and l. Line 8 stores the current state of the tree sampling on the variable AP. Then, if necessary, Line 9 removes symbols from γ_l and γ_r to guarantee their validity.

Using the non-terminals A,B,C, and the context γ_l,γ_r, Line 10 includes the rule $\gamma_l A\gamma_l \rightarrow \gamma_l BC\gamma_l$ in the rule list R. Line 11 calls the sampling algorithm recursively, passing as left-hand side the non-terminal B surrounded by the contexts γ_l and $C\gamma_{r-1}$, I as yield starting position, and $I+J$ as the yield ending position. Lines 12 and 13 adjust the contexts again. This adjustment is necessary because Line 11 has changed the current context.

Finally, Line 14 calls the sampling algorithm recursively, passing as left-hand side the non-terminal

C surrounded by the contexts γ_l and $C\gamma_{r-1}$, $I+J+1$ as yield starting position, and K as the yield ending position. At the end, the rules R will contain all sampled rules to yield w, representing a syntactic tree.

As the tree sampling in a P(k,l)CSG only allows left yields, there will never be a non-terminal to the left of w_i. Consequently, there will never be a terminal to the right of w_i while w_i is yielding. So, the contexts γ_l and γ_r, where $w = (w_i, w_{i+1}, ..., w_{j-1}, w_j), i < j$, are sequences containing only terminals and non-terminals respectively, that is, $\gamma \in \gamma_r \Rightarrow \gamma \in T$ and $\gamma \in \gamma_r \Rightarrow \gamma \in N$. Therefore, the probability of $A \in N$ yielding w_{ik}, given γ_l and γ_r, can by calculated by Equations 6 and 7.

$$P_{\gamma_l A \gamma_r, i, i} = \theta_{\gamma_l A \gamma_r \to \gamma_l w_i \gamma_r} \tag{6}$$

$$P_{\gamma_l A \gamma_r, i, k} = \sum_{\gamma_l A \gamma_r \to \gamma_l B C \gamma_r \in R} \sum_{j=i+1}^{k} \theta_{\gamma_l A \gamma_r \to \gamma_l B C \gamma_r} P_{\gamma_l B \gamma_{d-1}, i, j} P_{\gamma_l C \gamma_r, j+1, k} \tag{7}$$

To construct the inside table IT, we use the inside algorithm modified according to Equations 6 and 7. To infer a P(k,l)CSG with the GS, we need to run the GS algorithm with the sampling in Algorithm 2 and calculate IT for a P(k,l)CSG.

4 EXPERIMENTAL RESULTS

Given the algorithms described in Sections 3.2 and 3.3, we perform experiments to compare the GS and MH algorithms to understand the influence of the values (k,l) in a P(k,l)CSG.

4.1 Data Set

The *Billboard* data set (Burgoyne et al., 2011) contains 890 songs. Each file contains chord sequences split into labeled musical phrases. We extracted chord sequences from the data set *Billboard* to build our data set for inferences using the same procedure as in (Tsushima et al., 2018a): each word represents a chord sequence, and each chord represents a grammar symbol. After the extraction, we obtained a data set of 7217 words to infer our grammar. This data set allows us to extract the musical phrases to get words with a length smaller than the entire song. The advantage of this approach is to obtain a data set with a higher number of words. The source code for this work is available in our repository[2].

[2]https://github.com/henriqueblopes/grammatical-inferences

4.2 Inference Details

The total number of unique chords was 738. We used the same procedure as in (Tsushima et al., 2018a) to simplify the number $|N|$ of non-terminals N in our grammar. We transposed all phrases to C major and identified the 20 most frequent chords in the data set. We then switched the other chords to a terminal node referred to as "Other". We split the data set into a training data set TD with 80% of the phrases, and a validation data set VD with the other 20%.

Table 1 summarizes the combinations of factors in our experiment. Due to the high computational cost of GS, we limited the factors $|TD| \leq 3000$, and $|N| < 6$ in our experiments.

Table 1: Factors in our Experiment.

Algorithm	$\|N\|$	$\|TD\|$	k	l
MH	2	300	0	0
MH	2	300	1	0
MH	2	3000	0	0
MH	2	3000	1	0
MH	6	300	0	0
MH	6	300	1	0
GS	2	300	0	0
GS	2	300	1	0
GS	2	3000	0	0
GS	2	3000	1	0
GS	6	300	0	0
GS	6	300	1	0

We used the inference algorithms P(k,l)CSGs with GS described in Section 3.3 and MH (Shibata, 2014) with $k = \{0,1\}$ and $l = 0$. The GS with $(k,l) = (0,0)$ is one of the algorithms implemented in (Tsushima et al., 2018a). Therefore, we will compare them to the state of the art for the *Billboards* data set when performing our experiments. We run all algorithms in a Google Cloud virtual machine N1 series, High-CPU type, eight cores, 7.2GB of RAM, OS Debian GNU/Linux 10. The number of iterations was 200 and we limited the number of replicates to five for each combination of factors due to the high computational cost of GS inferences, as discussed in more detail in Section 4.4.

4.3 Inference Validation

We used the perplexity measure to validate the inferred grammar and used the cross-validation method dividing the data set into one different data sets per replicate. For each replicate, we calculated the average perplexity value (Equation 2) in the test set for each run.

4.4 Results

Figure 1 presents the box-plots for all values of perplexity in the y-axis for each replicate of the MH algorithm. Each pair of boxplots represents executions with $(k,l) = (0,0)$ and $(k,l) = (0,1)$.

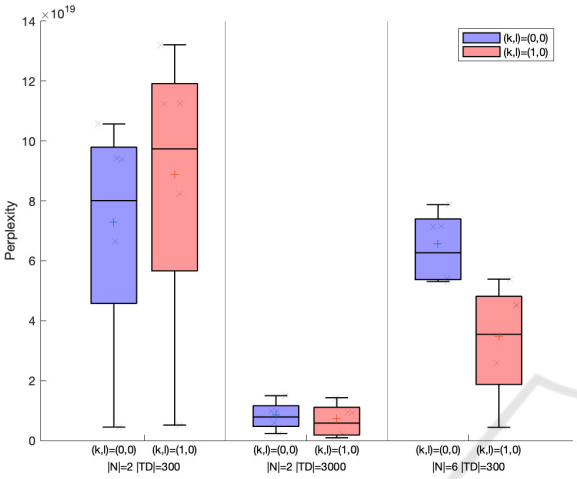

Figure 1: Perplexity for the MH algorithm. Experiments with a higher value of $|TD|$ and $|N|$ tend to achieve lower perplexity values. Also, grammars with $k = 1$ led to lower averages.

We can see that higher values of $|TD|$ and $|N|$ tend to achieve lower perplexity values. Also, when $N = 6$, the presence of a left context of length $k = 1$ led to a smaller perplexity value.

The GS samples a large number of trees at each iteration. So GS has a much higher run time than the MH algorithm, which is why we have a limited number of replicates. Figure 2 compares the run time of the GS and MH.

Figure 3 shows the perplexity values obtained with GS. In all cases, the P(k,l)CSGs have smaller average perplexity values than PCFGs.

As in the results for the MH algorithm, P(k,l)CSGs have smaller perplexity values than PCFGs using GS. The perplexity becomes smaller when we use a larger training set. This result also indicates the presence of the left context allowed the grammar to better represent the test sets.

In Figure 4, we compare the perplexity values for GS and MH. The results indicate GS can achieve lower perplexity values than MH.

We aggregated our results and used a Wilcoxon signed paired rank-test for a paired test for the null hypothesis that the average perplexity values for GS and MH are samples from distributions with equal medians. We obtained a p-value $= 0.0026$ indicating that

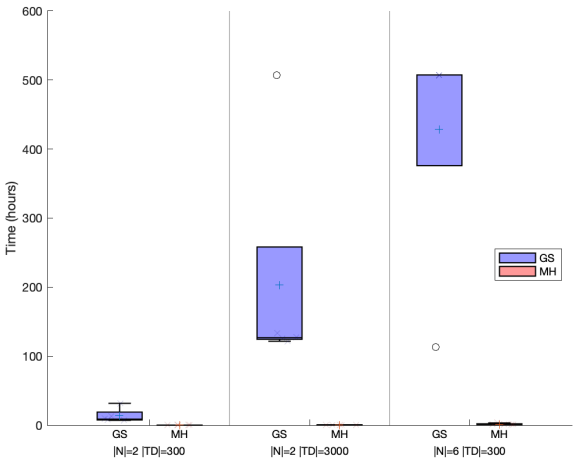

Figure 2: GS and MH run time.

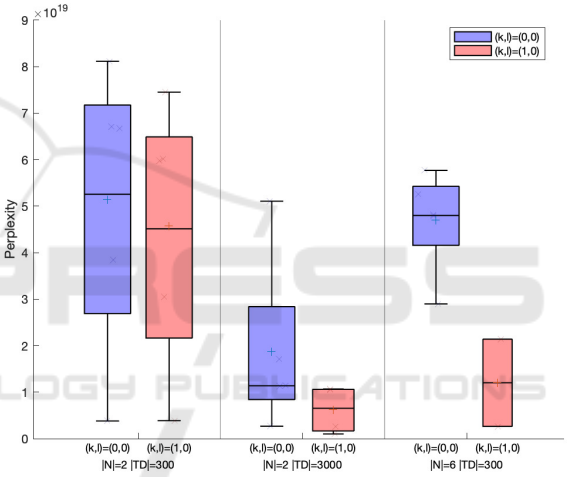

Figure 3: Perplexity Values for the GS algorithm. Grammars where $(k,l) = (1,0)$ tend to achieve lower perplexity values.

GS, despite its computational cost, can achieve lower median perplexity values than MH.

5 CONCLUSIONS

We have investigated the capability of P(k,l)CSG to represent features related to musical chord sequences present in the Billboards data set. We used Bayesian approaches to infer these grammars and implemented the first blocked GS algorithm to infer P(k,l)CSGs using the theory in (Shibata, 2014). Despite the high computational cost GS, the presence of a left context of size 1 provides smaller perplexity values than the absence of contexts to represent the test sets.

These findings mean we could use the inferred

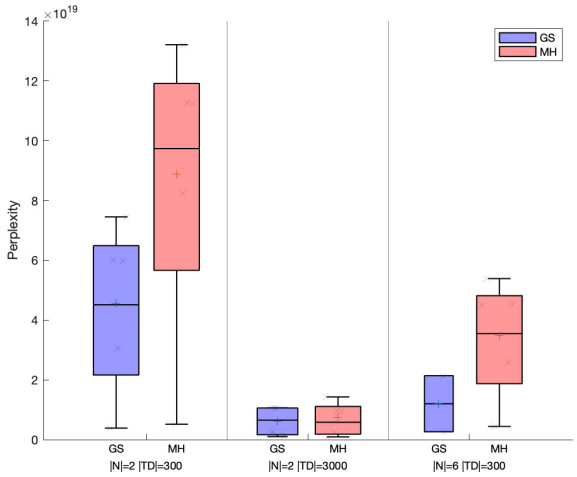

Figure 4: Perplexity Values for the GS and MH algorithms inferring P$(1,0)$CSGs.

models to perform machine learning tasks related to chord sequences like style classification or the generation of musical phrases. The data suggest we could improve our results by either investing in enough computational power or developing cheaper algorithms to infer grammars with $k > 1$ and $l > 0$.

ACKNOWLEDGMENTS

This work has been supported by the Brazilian Agencies State of Minas Gerais Research Foundation – FAPEMIG (APQ-00040-14); Coordination for the Improvement of Higher Level Personnel – CAPES; and National Council of Scientific and Technological Development – CNPq (402956/2016-8).

REFERENCES

Bhattacharjee, A. and Srinivasan, N. (2011). Hindustani raga representation and identification: A transition probability based approach. *International Journal of Mind, Brain and Cognition*, 2:71–99.

Blaß, M. (2013). Timbre-based drum pattern classification using hidden markov models. In *Machine Learning and Knowledge Discovery in Databases*.

Breuker, D., Matzner, M., Delfmann, P., and Becker, J. (2016). Comprehensible predictive models for business processes. *MIS Q.*, 40(4):1009–1034.

Burgoyne, J., Wild, J., and Fujinaga, I. (2011). An expert ground truth set for audio chord recognition and music analysis. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 633–638.

Forsyth, J. P. and Bello, J. (2013). Generating musical accompaniment using finite state transducers. In *Conference on Digital Audio Effects*, National University of Ireland.

Gagniuc, P. A. (2017). *Markov Chains: From Theory to Implementation and Experimentation.* Wiley.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741.

Heinz, J. and Rogers, J. (2010). Estimating strictly piecewise distributions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 886–896, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jarosz, G. (2013). Learning with hidden structure in optimality theory and harmonic grammar: beyond robust interpretive parsing. *Phonology*, 30(1):27–71.

Jarosz, G. (2015a). Expectation driven learning of phonology*.

Jarosz, G. (2015b). Learning opaque and transparent interactions in harmonic serialism.

Jäger, G. and Rogers, J. (2012). Formal language theory: Refining the chomsky hierarchy. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 367:1956–70.

Johnson, M., Griffiths, T., and Goldwater, S. (2007). Bayesian inference for pcfgs via markov chain monte carlo. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 139–146, Rochester, New York. Association for Computational Linguistics.

Kitani, K. and Koike, H. (2010). Improvgenerator: Online grammatical induction for on-the-fly improvisation accompaniment. In *Proceedings of the 2010 Conference on New Interfaces for Musical Expression*.

Mao, H., Chen, Y., Jaeger, M., Nielsen, T., Larsen, K., and Nielsen, B. (2012). Learning markov decision processes for model checking. *Electronic Proceedings in Theoretical Computer Science*, 103.

Nakamura, E., Nishikimi, R., Dixon, S., and Yoshii, K. (2018). Probabilistic sequential patterns for singing transcription. In *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1905–1912.

Nakano, M., Ohishi, Y., Kameoka, H., Mukai, R., and Kashino, K. (2014). Mondrian hidden markov model for music signal processing. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2405–2409.

Nouri, A., Raman, B., Bozga, M., Legay, A., and Bensalem, S. (2014). Faster statistical model checking by means of abstraction and learning. In Bonakdarpour, B. and Smolka, S. A., editors, *Runtime Verification*, pages 340–355, Cham. Springer International Publishing.

Shibata, C. (2014). Inferring (k,l)-context-sensitive probabilistic context-free grammars using hierarchical pitman-yor processes. In Clark, A., Kanazawa, M., and Yoshinaka, R., editors, *The 12th International Conference on Grammatical Inference*, volume 34 of

Proceedings of Machine Learning Research, pages 153–166, Kyoto, Japan. PMLR.

Takano, M. and Osana, Y. (2012). Automatic composition system using genetic algorithm and n-gram model considering melody blocks. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8.

Trochidis, K., Guedes, C., Anantapadmanabhan, A., and Klaric, A. (2016). Camel: Carnatic percussion music generation using n-gram models. In *Proceedings of the 13th Sound and Music Computing Conference*.

Tsushima, H., Nakamura, E., Itoyama, K., and Yoshii, K. (2017). Function- and rhythm-aware melody harmonization based on tree-structured parsing and split-merge sampling of chord sequences. In *ISMIR*.

Tsushima, H., Nakamura, E., Itoyama, K., and Yoshii, K. (2018a). Generative statistical models with self-emergent grammar of chord sequences. *Journal of New Music Research*, 47(3):226–248.

Tsushima, H., Nakamura, E., Itoyama, K., and Yoshii, K. (2018b). Interactive arrangement of chords and melodies based on a tree-structured generative model. In *ISMIR*.

Wang, C.-i. and Dubnov, S. (2017). Context-aware hidden markov models of jazz music with variable markov oracle. In *Proceedings of the Eighth International Conference on Computational Creativity*.