University of Oxford

UNIVERSITY OF OXFORD DEPARTMENT OF STATISTICS

A Bayesian non-parametric methodology for inferring grammar complexity

by

Achille Thin

Mansfield College

A dissertation submitted in partial fulfilment of the degree of Master of Science in Applied Statistics.

Department of Statistics, 24–29 St Giles,
Oxford, OX1 3LB

September 2019

This is my own work (except where otherwise indicated)

Candidate: Achille Thin

Signed:[illegible]

Date: September 8th, 2019

Abstract

Linguists have developed a theory classifying grammars by their complexity on a formal and computational way. Our goal here is, given a set of sentences from a specific language, to infer the complexity of the associated grammar. We develop in the following a Probabilistic Grammar Model that will be flexible enough to fit the sentences observed using a Sequential Monte Carlo Method. In the end, we want to perform model comparison between different classes of grammar, in the form of a Bayes Factor. Our model will be a Hierarchical Dirichlet Process, introduced in Yee Whye Teh and M.Blei (2016). The final motivation is to apply then this methodology to data from primates (Campbell's monkeys more precisely), to draw a meaningful conclusion on the complexity of their grammar (key problem already recognized as such in Jiang X (2018)).

In this dissertation, we will derive theoretical properties for the model, proving the relevance of the Bayes Factor in that case, and other conclusion on the behaviour of sentences produced by this type of models. A simulation study will also be presented to illustrate the theoretical points previously drawn.

Acknowledgements

I would like to thank Judith Rousseau, for her faith in me to present me with that project, and Robin Ryder, for his dedication and pedagogy. I have been honored by your trust during that dissertation, and it has been a pleasure to work with you, in Oxford or in Paris. I would like to thank you as well for the opportunity to publish a paper with you, I am very happy to keep on working on that project in the next weeks !

Finally, I would like to thank friends and family, who have shown a real interest in my work over the summer, and supported me during that time. In particular, I would like to thank Alvaro Herrera-Masurel, Ghislain and Sibylle Quinat, Florence, Mahaut Rey and my parents and sisters Christophe, Anne, Gabrielle and Marie.

Contents

List of Figures

1 Introduction

1.1 Motivation

Campbell's monkeys (*Cercopithecus campbelli*) can communicate through relatively evolved vocalizations, made of seven building blocks: *Boom, Hok, Hok-oo, Krak, Krak-oo, Wok, Wok-oo*. Different papers in linguistics journals, such as Schlenker et al. (2014), Ouattara et al. (2009) ask a series of relevant questions concerning the complexity of the primate's grammar, given that the sentences can be relatively long and elaborate :

- Boom Boom Hok-oo Hok-oo Hok-oo Hok-oo Hok-oo Krak-oo
- Hok Hok Hok-oo Krak-oo Krak-oo Wok-oo Wok-oo Wok-oo Krak-oo
- Hok Wok-oo Hok Krak-oo Krak-oo Krak-ooWok-oo Krak-oo Krak-oo Krak-oo Krak-oo Krak-oo Krak-oo Wok-oo Wok-oo Krak-oo Krak-oo Krak-oo Krak-oo

The complexity of a grammar can be assessed using Chomsky hierarchy (or Chomsky-Schutzenberger hierarchy) presented in Chomsky (1956) or Jager and Rogers (2012), which defines different classes for formal grammars, which we will develop in the latter.

The goal of this study is to construct a Bayesian Non Parametric Model for formal grammars, to model in particular Campbell's Monkeys language, and assess in the end the complexity of their grammar. It is important to note that the model presented in the following has already been developed by Lawrence Murray, Robin Ryder and Judith Rousseau. My contribution to that model relies on the Theoretical Results and Simulation Study parts of this dissertation. Moreover, this paper has been written in parallel of a paper that will be published in the next future with Lawrence Murray, Robin Ryder and Judith Rousseau, and thus any resemblance between this dissertation and the upcoming paper is completely intentional and deliberate.

1.2 Definition of a Formal Grammar

A formal grammar is defined by a set of rules over a given dictionary for a certain language. More formally, let us denote for a certain set X, X^* the set of sentences over X, *i.e.* $X^* = \{a_1...a_n; n \in \mathbb{N}, a_1, ..., a_n \in X\}$, $a_1...a_n$ being the concatenation of a_1, ..., a_n.

Let us also denote, for sets X and Y, $XY = \{ab; a \in X, b \in Y\}$.

Then, a formal grammar is defined as $G = (\mathcal{A}, \mathcal{B}, S, \mathcal{R})$, containing

- a finite set of terminal symbols $\mathcal{A} = \{a_1, \ldots, a_K\}$
- a finite set of non-terminal symbols $\mathcal{B} = \{B_0, \ldots, B_J\}$

- a distinguished non-terminal symbol B_0, the start symbol, often denoted by S in the literature
- a finite set of rules $\mathcal{R}$, each of the form $\alpha \to \beta$ where $\alpha \in \mathcal{B}$ (in context-free grammars - our interest here) and $\beta \in (\mathcal{A} \cup \mathcal{B})^*$.

In the following dissertation and model, we will only work in the Greibach Normal Form, introduced by Greibach in Greibach (1965). In this setting, all rules are of the form $\alpha \to \beta$ where $\alpha \in \mathcal{B}$ and $\beta \in \mathcal{A}(\mathcal{B})^*$, which means that any rule is of the form:

$$B \to aB_1...B_n$$

where B is a non terminal symbol, a a terminal symbol, and $B_1...B_n$ is a sequence of non terminals, possibly empty.

Note that any grammar considered in this dissertation can be equivalently written in the Greibach Normal form (Greibach (1965)), which validates our setting.

1.2.1 Production of a sentence from a grammar

To produce a sentence from a grammar G, start with the symbol S and then iteratively apply rules, until there are no non-terminals left. We present first an example before going further into detail of the production of a sentence.

Consider this simple grammar which produces a few English sentences, with $K = 4$, $\mathcal{A} = \{$the girl, an apple, an orange, eats$\}$ $a_1 =$"the girl", a_2="eats", $a_3 =$"an apple", $a_4 =$"an orange", $\mathcal{B} = \{B_0, B_1, B_2\}$, and $\mathcal{R}$ contains the elements

- $r_{0,1} : B_0 \to a_1B_1$
- $r_{0,2} : B_0 \to a_1B_1B_2$
- $r_{1,1} : B_1 \to a_2$
- $r_{2,1} : B_2 \to a_3$
- $r_{2,2} : B_2 \to a_4$

This grammar produces the following sentences:

- $a_1a_2 =$"the girl eats" (apply rules $r_{0,1}$ and $r_{1,1}$)
- $a_1a_2a_3 =$"the girl eats an apple" (apply rules $r_{0,2}$, $r_{1,1}$ and $r_{2,1}$)
- $a_1a_2a_4 =$"the girl eats an orange" (apply rules $r_{0,2}$, $r_{1,1}$ and $r_{2,2}$)

More formally, define the stack $\mathcal{X}_t$ of non terminal symbols representing all non terminal symbols left to treat. The stack of non terminal symbols $\mathcal{X}_t$ and the sentence y will be initialized by $\mathcal{X}_0 = (B_0)$ and $y_0 = []$. A rule is then applied to the non terminal at the top of the stack, which will add exactly one word (terminal) to the sentence - as we are using Greibach Normal form, and a random number of non-terminals to the stack. It goes on until there are no more non terminal symbols in the stack. The sentence produced is the last one observed.

Let us present a typical example.

- $y_0 = []$, $\mathcal{X}_0 = (B_0)$. We apply the rule $B_0 \to a_3B_1B_3$.
- $y_1 = [a_3]$, $\mathcal{X}_1 = (B_1B_3)$. We apply the rule $B_1 \to a_1B_2$.
- $y_2 = [a_3a_1]$, $\mathcal{X}_2 = (B_2B_3)$. We apply the rule $B_2 \to a_2$.
- $y_3 = [a_3a_1a_2]$, $\mathcal{X}_3 = (B_3)$. We apply the rule $B_3 \to a_1$.
- $y_4 = [a_3a_1a_2a_1]$, $\mathcal{X}_4 = ()$. The stack is empty, we stop here.

The sentence produced is thus $y = [a_3a_1a_2a_1]$.

This presents the production of a sentence from a grammar. It is important to notice that for each latent variable B_i, we have a set of rules. Only one of those rules will be at each step, as presented above.

1.2.2 Hierarchy of the formal grammars

The Chomsky hierarchy (presented in Chomsky (1956)) can be summarized in the following figure:

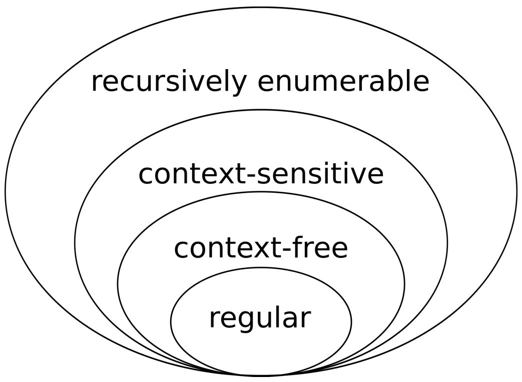

Figure 1: Chomsky Hierarchy for formal grammars

We can see here that the different classes are nested into another. They go from the least complex, the regular class, to the most, the recursively enumerable. The classes have also been described in Jager and Rogers (2012), which presents a refinement of Chomsky Hierarchy. The human languages are usually between the context-sensitive class and the context free, but we will focus here on the context-free and regular classes. Proving then that Campbell's monkeys grammar belongs at least to the context-free class would then be a big result.

In the Greibach Normal Form, a regular grammar has production rules of the form $B \to aC$ where $B, C \in \mathcal{B}$, $a \in \mathcal{A}$ or $B \to a$ where $B \in \mathcal{B}$, $a \in \mathcal{A}$. The length of a rule is then either 0 or 1. This looks like a Hidden Markov Model structure, where the hidden state is the latent variable in $\mathcal{B}$, and the emission at each step is a word from $\mathcal{A}$. One of the key features here, like in Hidden Markov Models, is that there is no possibility for long term memory.

A context-free grammar, on the other hand, has production rules of unbounded length, *i.e.*, in Greibach Normal Form, of the form $B \to aB_1...B_n$ where $n \in \mathbb{N}$, (n can be equal to 0), $B, B_1, ..., B_n \in \mathcal{B}$, $a \in \mathcal{A}$. This produces a tree, where each node can be associated with a non terminal symbol in $\mathcal{B}$, and each leaf to a word $a \in \mathcal{A}$.

Our problem here is simple to state. We are given a set of sentences (produced by Campbell's monkeys for the prior motivation, for example), with the hypothesis that they are issued from a formal grammar $\mathcal{G}$. The goal is to decide which class of grammars $\mathcal{G}$ belongs to, *i.e.* to perform the test :
H_0: "$\mathcal{G}$ is a regular grammar" versus H_1: "$\mathcal{G}$ is a context-free grammar".

1.3 Definition of a Probabilistic Grammar

We can now define a probabilistic extension to our formal regular and context-free grammars. For a formal grammar $G = (\mathcal{A}, \mathcal{B}, B_0, \mathcal{R})$, we denote as $\mathcal{R}_j$ the set of rules which start with B_j, *i.e.* $\mathcal{R}_j = \mathcal{R} \cap \{(B_j \to \beta) : \beta \in \mathcal{A}\mathcal{B}^*\}$, the rules that can be applied when B_j is at the top of the stack. If we define a probability distribution P_j over each of the $\mathcal{R}_j$, we obtain a probabilistic grammar.

We define thus a probabilistic grammar as $G = (\mathcal{A}, \mathcal{B}, B_0, \mathcal{R}, \mathcal{P})$, where $(\mathcal{A}, \mathcal{B}, B_0, \mathcal{R})$ is a formal grammar, and let $J + 1 = |\mathcal{B}|$ be the number of non terminal symbols. $\mathcal{P} = (P_0, \ldots, P_J)$ is such that each P_j ($j = 0 \ldots J$) is a probability distribution on $\mathcal{R}_j$.

To define such grammar, we will use in the following the acronym PCFG, which means Probabilistic Context-Free Grammars, our field of study here.

2 Presentation of the Model for Probabilistic Context Free Grammars

We now want to model a probabilistic context free grammar $G = (\mathcal{A}, \mathcal{B}, B_0, \mathcal{R}, \mathcal{P})$. The set $\mathcal{A}$ is here given and finite. The set $\mathcal{B}$ of latent variables is assumed random in the modelling, and infinite to avoid bounding it by a pre-determined number of distinct non terminal symbols. We can then be sure that our model is non-parametric. We thus have that $\mathcal{R}$ and $\mathcal{P}$ must be infinite as well, and random. We will use Dirichlet Processes (introduced by Ferguson (1973)) for the modelling of those infinite random sets, denoted in the following as $DP(M, H)$, where $M > 0$, and H is a probability distribution on a given measurable set.
In the following, we will use alternatively the Sethuraman representation of the Dirichlet process (Sethuraman (1994)) or the equivalent form of the Chinese Restaurant Process, D.J. (1985) (variant of the also equivalent *Polya Urn* representation, Blackwell and MacQueen (1973)).

2.1 Chinese Restaurant Process and Sethuraman Representation

The Sethuraman representation (Sethuraman (1994)) is also called the *stick-breaking construction*. To model the Dirichlet Process $Q \sim DP(M_1, H_0)$, where H_0 is a probability distribution on $\mathbb{R}$., we sample

$$V_i \overset{iid}{\sim} \text{Beta}(M_1, 1) \text{ for } i \in \mathbb{N}^*, \quad b_j \overset{iid}{\sim} H_0 \text{ for } j \in \mathbb{N}^*$$

Then the Sethuraman representation (Sethuraman (1994)) defines Q the random measure such that

$$q_j = V_j \prod_{i<j} (1 - V_i) \,, \quad Q = \sum_{j=1}^{\infty} q_j \delta_{(b_j)}$$

Where $\delta_{(b_j)}$ is the probability measure putting all its mass on b_j.

Let us remember the Chinese Restaurant Process (D.J. (1985)). It is a discrete-time stochastic process, parametrized by a base measure, here noted as H and a concentration parameter, here noted as M. It is non parametric as it can have an infinite number of tables (here, the features), and goes recurrently like this.

- The first customer (Customer 1) sits at table 1
- After n customers, suppose there are p tables occupied, and denote the number of customers at table $1 \leq k \leq p$ by $c_k^{(n)}$. Then Customer $n + 1$ chooses table k with probability
$$\frac{c_k^{(n)}}{n + M}$$
or a new table (which will be denoted as table $p + 1$ afterwards) with probability
$$\frac{M}{n + M}$$

.

One of the key features of the Chinese Restaurant process is that even if it is non parametric and an infinite number of tables will be occupied almost surely, it has a "riches get richer" property (the more you are at a table, the more probably you are chosen for the next customer). That way, we can approximate quite effectively the Process, stopping at a given number of tables (which will be increasing with the goodness of our approximation).

2.2 Formal Model

Note that in the Greibach Normal form, a grammar rule for B_i is determined by a word a in $\mathcal{A}$, a number of non terminal symbols L (in $\{0,1\}$ for regular grammars, or in $\mathbb{N}$ for context free grammars), and the non terminal symbols associated $B_1, ..., B_L \in \mathcal{B}$.

Let $\mu_{\mathcal{A}}$ be a probability distribution on $\mathcal{A}$ and H_L a probability on the set of non negative integers $\mathbb{N}$.

The probabilistic model is then defined by the following hierarchy:

- (i) Model for $\mathcal{B}$: $B_0 = 0$ and let $Q \sim DP(M_1, H_0)$, where H_0 is a probability distribution on $\mathbb{R}$. Then the Sethuraman representation (Sethuraman (1994)) Q means that
$$Q = \sum_{j=1}^{\infty} q_j \delta_{(b_j)}, \quad q_j = V_j \prod_{i<j}(1-V_i), \quad V_i \overset{iid}{\sim} \text{Beta}(M_1, 1), \quad b_j \overset{iid}{\sim} H_0$$
In other words $\mathcal{B} = \{B_0, b_j, j \geq 1\}$.
- (ii) Model for $\mathcal{P}$: The probabilistic model for rules $R_B : B \to aB_1 \cdots B_L$ is described as: For all $B \in \mathcal{B}$ generate independently $P_B \sim DP(M_2, H_P(\cdot|Q))$, where $H_P(\cdot|Q)$ is a probability distribution on $\mathcal{A} \times \cup_{n=0}^{\infty} \mathcal{B}^n$ defined by:
$$H_P(\beta = (a, B_1 \cdots B_L)|Q) = \mu_{\mathcal{A}}(a) H_L(L) \left(\prod_{j=1}^{L} Q(B_j) \right)^{\mathbf{1}_{L>0}}, \tag{1}$$
The rules R_b (in the form $b \to \beta$) are then generated by:
$$R_b^i | P_b \overset{iid}{\sim} P_b, i \geq 1$$

Conditionnally on Q, we can integrate out the P_b's and obtain the marginal distribution of the rules $R_{t,B_t} : B_t \to \beta_t$, $t = 1, \cdots, n$, which are described by independent Chinese Restaurant Processes (D.J. (1985)): denote for each B, $\mathcal{I}_t(B) = \{i \leq t; B_i = B\}$ where B_i here refers to the latent state in the rule R_{i,B_i} and $N_t(B)$ the cardinality of $\mathcal{I}_t(B)$, then if $B_t = B$

$$\begin{aligned}\mathbb{P}_{B_t=B}\left(\beta_t | B_t = B, R_{i,B_i}, i \leq t-1\right) &= \mathbb{P}_{B_t=B}\left(\beta_t | B_t = B, \beta_i, i \in \mathcal{I}_{t-1}(B)\right) \\ &= \frac{M_2}{M_2 + N_{t-1}(B)} H_P(\beta_t|Q) + \frac{1}{M_2 + N_{t-1}(B)} \sum_{i \in \mathcal{I}_{t-1}(B)} \delta_{(\beta_i)}\end{aligned} \tag{2}$$

We can then further integrate out Q, which is called only in 2 when a rule is sampled from H_P. We can rewrite (2) as

$$Z_t \sim \text{Ber}\left(\frac{M_2}{M_2 + N_{t-1}(B)}\right)$$

If $Z_t = 1$ then $\beta_t \sim H_P(\beta_t|Q)$ else

$$\beta_t \sim \frac{1}{N_{t-1}(B)} \sum_{i \in \mathcal{I}_{t-1}(B)} \delta_{(\beta_i)}.$$

Let $\mathcal{Z}_t = \{i \leq t; Z_i = 1\}$ and $\mathcal{B}(\mathcal{Z}_t) = \cup_{i \in \mathcal{Z}_t} \mathcal{B}_{\beta_i}$ where $\mathcal{B}_{\beta_i}$ is the set of latent nodes appearing in β_i (possibly non distinct), i.e. if $\beta_i = (a_i, L_i, B_1, \cdots, B_{L_i})$, then $\mathcal{B}_{\beta_i} = \{B_1, \cdots, B_{L_i})\}$. Note that if a value B appears twice (or more) in the collection of sets $\mathcal{B}_{\beta_i}$, $i \leq t$, then it appears twice (or more) in $\mathcal{B}(\mathcal{Z}_t)$.

Then the marginal distribution of $(R_{t,B_t}, Z_t, t \leq n)$ is given by the following conditional rule : if $B_t = B$ and $R_{t,B_t} : B \to \beta_t$, then

$$Z_t \sim \text{Ber}\left(\frac{M_2}{M_2 + N_{t-1}(B)}\right)$$

if $Z_t = 0$ then

$$\beta_t \sim \frac{1}{N_{t-1}(B)} \sum_{i \in \mathcal{I}_{t-1}(B)} \delta_{(\beta_i)},$$

else $\beta \sim H_P(\beta_t|\mathcal{B}(\mathcal{Z}_t))$ defined by

$$H_P(\beta_t = (a_t, L_t, B_1^t, \cdots, B_{L_t}^t)|\mathcal{B}(\mathcal{Z}_t)) = \mu_{\mathcal{A}}(a_t) H_L(L_t) \mathbb{Q}((B_1^t, \cdots, B_{L_t}^t)|\mathcal{B}(\mathcal{Z}_t))$$

where $\mathbb{Q}$ is the Chinese restaurant process associated to the model $B_i|Q \overset{iid}{\sim} Q$, and $Q \sim DP(M_1, H_0)$ so that

$$\mathbb{Q}(dB_n|B_1, \cdots, B_{n-1}) = \frac{M_1}{M_1 + n - 1} H_0(dB_n) + \frac{1}{M_1 + n - 1} \sum_{i=1}^{n-1} \delta_{(B_i)}(dB_n). \quad (3)$$

For any finite set of sentences $\mathcal{Y}$, there always exists an infinite number of regular grammars which generate $\mathcal{Y}$, but these might be extremely complex (large number of non-terminals and/or rules). To perform our model choice, we therefore need to define a probability distribution over the set of all Probabilistic Context Free Grammars, which penalizes grammars with more non-terminals and more rules.

Since we wish the number of non-terminals and rules to be unbounded, but to penalize large numbers, a natural model is the Chinese Restaurant Process.

We now construct a probabilistic context free grammar based on Greibach normal form by permitting the set of non-terminals, $\mathcal{B}$, and set of rules, $\mathcal{R}$, to be of countably-infinite size, and providing a suitable probabilistic model over them. Although these sets are infinite, the number of non-terminals and rules used to generate a finite set of finite sentences will necessarily be finite.

2.3 Process for rules

For each non-terminal B_j, we have a set of rules $\mathcal{R}_j$. Each of these rules come from a Chinese Restaurant Process, meaning there is one Chinese Restaurant Process per non-terminal. When we need to process non-terminal B_j, we generate a rule from the Chinese Restaurant Process C_j. The n_j-th time that B_j is on the stack, we pick existing rule r_{ji} with probability $\frac{n_{ji}}{n_j+\theta}$, and we create a new rule with probability $\frac{\theta}{n_j+\theta}$.

2.4 Creating a new rule

When we need to create a new rule, it will be of the form $r_{ji} : B_j \to a_k B_{k_1} B_{k_2} \ldots B_{k_l}$. The process to create a rule is:

- Draw $a_k \sim Categorical(\mathcal{A})$ (with Dirichlet prior on probabilities).
- Draw $l \sim Poisson(\lambda)$.
- Draw sequentially $B_{k_1}, \ldots, B_{k_l}$ from a Chinese Restaurant Process $\tilde{C}$.

At each step, there is therefore a positive probability of generating one or several new non-terminals which had never been observed before.

If we wish to generate a regular grammar instead of a PCFG, we need to impose $l \in \{0, 1\}$, so we replace the step $l \sim Poisson(\lambda)$ by $l \sim Bernoulli(p)$.

We have a hierarchy of Chinese Restaurant Processes: the Chinese Restaurant Process $\tilde{C}$ which generates non-terminals, and for each non-terminal B_j, the Chinese Restaurant Process C_j which generates rules in $\mathcal{R}j$.

This process will be the one used in the simulations for the length, at the end of this dissertation.

2.5 Method for assessing the complexity

The idea, from those models, is to compute the different PCFG, with probability distributions for the length of the rules being different in the regular or the context free class.

For regular grammar, all rules are of length 0 or 1, hence the probability distribution H_L on $\mathbb{N}$ will be a Bernoulli, of parameter p a given hyperparameter of the model.

For Context free grammars, the length of rules is not bounded, hence H_L will typically be a Poisson Law, of parameter λ, which we will further investigate in the simulation study part.

Once those models are settled, we can compute in each case a marginal likelihood for each model, given a set of sentences, by exploring a number of potential grammars in under each hypothesis.

Using Sequential Monte-Carlo method, we can then compute a Bayes Factor to draw conclusions about our problem. However, the simple idea of a Bayes Factor has no sense if the probability of hypothesis H_0 under H_1, $\mathbb{P}_{H_1}(H_0)$, is not equal to 0.

This question is not evident at all, considering it is not clear which context-free grammar can be expressed in terms of regular grammars (There is no simple criterion that allows us to conclude that a context-free grammar can or cannot be expressed in terms of a regular grammar).

This will be the main contribution (proving that the Bayes Factor is a reasonable tool for model choice).

We will afterwards do a study of the probability that a sentence drawn from this process is finite.

3 Theoretical Results

3.1 Proof of the relevance of the Bayes factor

Consider the set $\overline{\mathcal{G}}$ of grammars such that almost surely, if $G = (\mathcal{A}, \mathcal{B}, B_0, \mathcal{R}, \mathcal{P}) \in \overline{\mathcal{G}}$, then $B \in \mathcal{B} \Rightarrow \mathbb{E}_{P_B}(L_B) \leq 1$, where L_B is the function which associates a rule from R_B to its length. $\overline{\mathcal{G}}$ is thus the set of grammars with rules length expectation all inferior or equal to 1, for each non-terminal symbol. We have obviously all regular grammar in $\overline{\mathcal{G}}$, as the length of each rule can only be 0 or 1.

We will compute then $\mathbb{P}_{H_1}(\forall i, \mathbb{E}[L_i] \leq 1 | (P_{B_i})_{i \in \mathbb{N}})$, where P_{B_i} is the Dirichlet process for the rules issued by the i-th latent variable B_i, and L_i the random variable modelling the length of a rule drawn from P_{B_i}, *i.e* $\mathbb{P}_{H_1}(\overline{\mathcal{G}})$ in our model.

We will denote $\{P_{B_i}, i \in \mathbb{N}\}$ as $\mathcal{P}$, as noted before.

First, let us notice that conditionally on P_{B_i} the Dirichlet process for the rules issued by the i-th latent variable B_i, L_i the length of the rule drawn from P_{B_i} is independent to all other $(P_{B_j})_{j \neq i}$.

Hence, $\mathbb{P}(\mathbb{E}(L_i) \leq 1 | P_{B_i}) = \mathbb{P}(\mathbb{E}(L_i) \leq 1 | (P_{B_i})_{i \in \mathbb{N}})$, by conditional independence (as $\mathbb{E}(L_i) \leq 1 | P_{B_i}$ is deterministic).

It is also important to note that $\mathbb{P}(\mathbb{E}(L_i | \mathcal{P})) = \mathbb{P}(\mathbb{E}(L_i) | \mathcal{P})$: Indeed, as $\mathbb{E}(L_i)$ is a determined function given $\mathcal{P}$. We will thus in the following use either of the form, according to which is the clearer at the moment.

Let us denote A_i the event $A_i = \{\mathbb{E}(L_i) \leq 1 | (P_{B_i})_{i \in \mathbb{N}}\}$.

We have $\limsup(A_i) = \{\forall i, \mathbb{E}(L_i) \leq 1 | (P_i)_{i \in \mathbb{N}}\}$.

Given P_{B_i}, the distribution of L_i is known, as L_i is just the marginalisation on the length of the rule issued from P_{B_i}. It stays known given all the $(P_{B_i})_{i \in \mathbb{N}}\}$

We thus have that $\mathbb{E}(L_i | (P_{B_i})_{i \in \mathbb{N}}\})$ which is deterministic, as the distribution of L_i is known given P_{B_i}.

Then, the events $\{\mathbb{E}(L_i | (P_{B_i})_{i \in \mathbb{N}}\}) \leq 1\}$ and $\{\mathbb{E}(L_j | (P_{B_i})_{i \in \mathbb{N}}\}) \leq 1\}$ for $i \neq j$ are independent, as different deterministic events.

From there, we can apply Borel-Cantelli's lemma for independent events which state that their $\limsup$ is either 0 or 1.

In H_1, let i be in $\mathbb{N}$. The probability of drawing the first rule in P_{B_i} to be of probability superior to $3/4$ is strictly positive for any $\alpha \in \mathbb{R}_*^+$ constant of the Dirichlet Process associated with the rules. And the probability for drawing the length of that rule to be superior to 2 is also strictly positive, for any $\lambda \in \mathbb{R}$ constant of the Poisson distribution for the length of the rules. Hence, the event $\{\mathbb{E}(L_i) \geq \frac{3}{4} \times 2 = \frac{3}{2}\}$ has a probability strictly superior to zero, and $\mathbb{P}(\mathbb{E}(L_i) \leq 1 | (P_{B_i})_{i \in \mathbb{N}}) < 1$.

We thus have $\mathbb{P}(\limsup(A_i)) = \mathbb{P}(\forall i, \mathbb{E}(L_i) \leq 1 | (P_{B_i})_{i \in \mathbb{N}}) < 1$, and thus

$$\mathbb{P}(\limsup(A_i)) = \mathbb{P}(\forall i, \mathbb{E}(L_i) \leq 1 | (P_{B_i})_{i \in \mathbb{N}}) = 0$$

As we had $\mathbb{P}_{H_1}(\forall i, \mathbb{E}(L_i) \leq 1 | (P_{B_i})_{i \in \mathbb{N}}) \geq \mathbb{P}_{H_1}(H_0)$, we have in the end $\mathbb{P}_{H_1}(H_0) = 0$.

We thus have that the Bayes factor is relevant, and our method is applicable !

Moreover, we can see that by the same reasoning, we can catch more results on the behaviour of our rules under our prior model, which are described in the next lines.

On the same way, under H_1, for any $m \in \mathbb{R}^+$ $\mathbb{P}(\forall i, \mathbb{E}(L_i) \leq m | (P_{B_i})_{i \in \mathbb{N}}) = 0$.We thus have that almost surely, the L_i are not bounded.

Let us prove that $\mathbb{P}(\forall i, \mathbb{E}(L_i) \leq +\infty | (P_{B_i})_{i \in \mathbb{N}}) = 1$.

Let i be in $\mathbb{N}$, m be in $\mathbb{R}$.

By Markov inequality, we have:

$\mathbb{P}(\mathbb{E}(L_i|\mathcal{P}) \leq m) \leq \frac{\mathbb{E}(\mathbb{E}(L_i|\mathcal{P}))}{m}$

Moreover, we know that the marginal expectation for the L_i is the expectation of the Poisson law for the length of the rule, hence,

$\mathbb{P}(\mathbb{E}(L_i|\mathcal{P}) \leq m) \leq \frac{\lambda}{m}$

Taking m to infinity, we do have in the end $\mathbb{P}(\mathbb{E}(L_i|\mathcal{P}) = +\infty) = 0$

3.2 Galton-Watson Trees and Generalization

For the next section, we introduce an alternative representation of the model, with Galton Watson Trees.

A Galton-Watson tree (described fully in Abraham and Delmas (2015)) is a simple model for a branching population. It was originally introduced by Francis Galton to investigate the propagation of family names, on the following baseline:

Let us suppose each adult male transmits his family name to his children. The name is surviving at the next generation if one of the children carrying it is a male (the offspring is randomly chosen to be male of female). Else, if there is no male descendant, the family name dies. One of the key assumption here is that the distribution of the offspring is the same for all individuals along the process.

The study of that process allows us to define a probability of survival of the family name, which was a key concern in the mid 19-th Victorian England when aristocratic families were afraid their family name could become extinct. They thus asked Sir Francis Galton (1873,Educational Times): *How many male children (on average) must each generation of a family have in order for the family name to continue in perpetuity?*, which was answered in 1874 by his paper *One the probability of extinction of families* with the introduction of this model (Watson and Galton (1875)).

More formally, let us describe the model as presented by Abraham and Delmas (Abraham and Delmas (2015)).

3.2.1 Formal Representation of a Monotype Galton Watson Tree

Let ζ be a random variable in $\mathbb{N}$ distributed according to $p = (p(n))_{n\in\mathbb{N}} : p(n) = \mathbb{P}(\zeta = n)$. Let $m = \mathbb{E}(\zeta)$ the expectation of ζ and g its generating function : $g(r) = \sum_{i\in\mathbb{N}} p(i)r^i = \mathbb{E}(r^\zeta)$.

The evolution of the Galton Watson process $(Z_n)_{n\in\mathbb{N}}$ with offspring distribution p is defined as follows:

- Z_n is the size of the population at the time n. Initially, we have Z_0 = 1.
- The size of the population at time $n + 1$ given the population at time n is obtained by summing all the children of each individual alive at time n. Each individual has a random number of children, sampled from the distribution p - thus identically distributed as ζ - and all independent from each other. Each individual alive at the generation n dies after giving birth (each individual has only one generation of children).

Mathematically it can be written: (with the convention that $\sum_{i=1}^{0} = 0$): Set $Z_0 = 1$ and for $n \in \mathbb{N}^*$: $Z_n = \sum_{i=1}^{Z_{n-1}} \zeta_i^{(n)}$ Where $(\zeta_i^{(n)})_{i,n \in \mathbb{N}}$ are independent realisations of p, thus distributed as ζ.

The extinction event can be defined by $\mathcal{E} = \{\exists n \in \mathbb{N}; Z_n = 0\}$.

Now, the key question is, considering a given offspring distribution p, to compute the extinction probability $\mathbb{P}(\mathcal{E})$.

Following Abraham and Delmas (2015), we define the distribution to be

- critical, if $m(p) = \mathbb{E}(\zeta)$ where ζ is distributed according to p is equal to 1 $m(p) = 1$.
- sub-critical if $m(p) < 1$.
- super-critical if $m(p) > 1$.

3.2.2 Computation of the Extinction Probability

This subsection presents results from Abraham and Delmas Abraham and Delmas (2015) on extinction probability in a mono-type Galton Watson Trees case.

A few particular cases arise, when studying the extinction probability.

First, if $p(0) = 0$, then almost surely we have the survival of the population at each step, hence $\mathbb{P}(\mathcal{E}) = 0$.

If $p(0) = 1$, we have extinction almost surely and $\mathbb{P}(\mathcal{E}) = 1$.

Finally, if $p(0) + p(1) = 1$, we have the Galton Watson process which can be modeled as the realisation of a geometric variable (Here, the total height of the tree), of parameter $p(1)$. Then, either $p(1) = 1$ and the tree is just an infinite spine almost surely and $\mathbb{P}(\mathcal{E}) = 0$, or $p(1) < 1$ and we have $\mathbb{P}(\mathcal{E}) = 1$.

Those cases treated, let us suppose now that $0 < p(0) < 1$ and $p(0) + p(1) < 1$.

One can notice that under those conditions, the generating function g is strictly convex.

Let us prove first that the probability of extinction must be a fixed point of g.

For that, let us take a tree τ with root u, and denote, for any vertex v in τ, $k_v(\tau)$ the size of the offspring of v in τ.

We have
$$\mathbb{P}(\mathcal{E}) = \mathbb{P}(\mathcal{E}(\tau)) = \sum_{k\in\mathbb{N}} \mathbb{P}(\mathcal{E}(\tau_{u_1}), ..., \mathcal{E}(\tau_{u_k})|k_u(\tau) = k)p(k)$$
by the law of total probability, denoting τ_{u_i} the Galton Watson tree having as a root the i-th children of u the root of τ (for $i \leq k_u(\tau)$).

By the properties of Galton Watson trees, the events $\mathcal{E}(\tau_{u_1}), ..., \mathcal{E}(\tau_{u_k})$ are independent given $k_u(\tau) = k$. Moreover, by the identically distributed property of all offsprings, we have, for any $i \leq k_u(\tau)$, $\mathbb{P}(\mathcal{E}(\tau_{u_i})|k_u(\tau) = k) = \mathbb{P}(\mathcal{E})$. We can thus write
$$\mathbb{P}(\mathcal{E}) = \sum_{k\in\mathbb{N}} \mathbb{P}(\mathcal{E})^k p(k) = g(\mathbb{P}(\mathcal{E}))$$
And thus, $\mathbb{P}(\mathcal{E})$ is a fixed point for g.

Now, remember that under the conditions described before, g is strictly convex.

We thus have two different cases.

If g is sub-critical or critical, we have $g'(1) \leq 1$, and $g(1) = 1$. We thus have that $g'(x) < 1, x < 1$. It means that the function $h = g - Id$ is monotonous over $[0, 1]$ (and strictly decreasing on $[0, 1)$). That means that h can be equal to 0 at most once on $[0, 1]$ and thus 1 is the only fixed point of g on $[0, 1]$ in that case.

Therefore, the probability of extinction $\mathbb{P}(\mathcal{E})$ of the tree τ is equal to 1 in the critical and sub-critical cases.

If g is super-critical, there will be two fixed points for g in $[0, 1]$, 1 and $q \in [0, 1)$. We will prove that $\mathbb{P}(\mathcal{E}) = q$. Let $\mathbf{t}$ be a certain finite determined tree. Let τ be a Galton Watson tree, with offspring distribution p (with generating function g), and consider the distribution
$$\tilde{p}(n) = q^{n-1}p(n)$$
We have that $\tilde{p}$ is a distribution, as $\sum_{n\in\mathbb{N}} \tilde{p}(n) = \sum_n q^{n-1}p(n) = g(q)/q = 1$, and the Galton Watson tree associated $\tilde{\tau}$ is sub-critical :
$$\forall r \in [0, 1], \tilde{g}(r) = \sum_{n\in\mathbb{N}} r^n q^{n-1}p(n) = \frac{g(rq)}{q}$$
and thus $\tilde{g}'(r) = g'(rq)$, hence $\tilde{g}'(1) = g'(q) < 1$ and $\tilde{\tau}$ is sub-critical.

Finally, we can write, for any finite determined tree $\mathbf{t}$,
$$q\mathbb{P}(\tilde{\tau} = \mathbf{t}) = q\prod_{u\in\mathbf{t}} p(k_u(\mathbf{t}))q^{k_u(\mathbf{t})-1} = q^{1+\sum_{u\in\mathbf{t}}(k_u(\mathbf{t})-1)} \prod_{u\in\mathbf{t}} p(k_u(\mathbf{t})) = \mathbb{P}(\tau = \mathbf{t})$$
as we have $\mathbb{P}(\tau = \mathbf{t}) = \prod_{u\in\mathbf{t}} p(k_u(\mathbf{t}))$ and $\sum_{u\in\mathbf{t}} k_u(\mathbf{t}) = |\mathbf{t}|-1$, so $1+\sum_{u\in\mathbf{t}} (k_u(\mathbf{t}) - 1) = 0$.

Summing over all finite trees, we have
$$\mathbb{P}(\mathcal{E}(\tau)) = q\mathbb{P}(\mathcal{E}(\tilde{\tau})) = q$$

as $\tilde{\tau}$ has a sub-critical offspring distribution.

In the case where all individuals in the Galton Watson Tree have the same offspring distribution, we have a clear criterion to determine if a tree is finite or not, and if not, with what probability it will become extinct (*i.e*, converge).

Let us now consider a multi type setting, more appropriate in our case, where each of the non-terminal symbols B_i has its own offspring distribution.

3.2.3 Generalization on a multi-type Galton Watson Tree

In that subsection, we propose a novel generalization of the computation of the extinction probability in a multi-type Galton Watson tree (with a finite number of types). Studies of the sort have been presented in Abraham and Delmas (2016) or Seneta (1970), but no clear generalization of the fixed point method have been presented. This will be our goal here. The study has indeed been performed so far under the hypothesis that all individuals have the same offspring distribution. However, in our case, an individual represents a latent variable B_i so that each different individual has a specific offspring distribution P_{B_i}.
We thus have to introduce and generalize that study for multi-type trees.

In the following, we will suppose that there are at most D types of individuals (in the Chinese Restaurant Process setting, it just means that only D tables have been discovered).

Each type i of individuals will have the same offspring distribution P_i. A realization ζ_i of P_i will be a vector of $\mathbb{R}^D$:
$\zeta_i = \big(\zeta_i(1)\zeta_i(2)...\zeta_i(D)\big)$ where $\zeta_i(j)$ represents the number of children of type j of ζ_i.

We will denote as well, at the n-th generation, the vector $\underline{Z_n} = \big(Z_n(1)...Z_n(D)\big)$ to be the vector of all individuals alive, $Z_n(i)$ denoting the number of individuals alive at generation n of type i for $i \in 1, ..., D$.

To define properly this multi-type Galton Watson process as before, we can write:

- $\underline{Z_0} = \big(1, 0, ...0\big)$ the vector with a 1 on the first line (for the presence of the start symbol only in the stack) and zeros elsewhere.
- On the same way, at generation n+1, the generation n dies by giving birth to their offspring, only this time the offspring will be separated into the different types.

We can thus write mathematically, as before: $\underline{Z_0} = \big(1, 0, ...0\big)$ and for $n \in \mathbb{N}^*$,

$$\underline{Z_n} = \sum_{i=1}^{D} \sum_{t=1}^{Z_{n-1}(i)} \zeta_i^{(t)} \tag{4}$$

where, for each i in $\{1, ..., D\}$, the $(\zeta_i^{(t)})_t$ are independent identically distributed random vectors according to P_i. This can be written coordinates by coordinates, considering that the $(\zeta_i^{(t)})_{i,t}$ are vectors of $\mathbb{R}^D$.

$$Z_n(j) = \sum_{i=1}^{D} \sum_{t=1}^{Z_{n-1}(i)} \zeta_i^{(t)}(j) \qquad \forall j \in \{1, ..., D\}$$

Let us now remember the expression of a generating function for a random vector in $\mathbb{N}^D$. If $\underline{\zeta} = \big(\zeta(1), ...\zeta(D)\big)$ is a random vector in $\mathbb{N}^D$ with probability distribution $(p(n_1, ..., n_D), (n_1, ..., n_D) \in \mathbb{N}^D)$, we have its generating function:

$$g(r_1, ...r_D) = \mathbb{E}(r_1^{\zeta_1} ... r_D^{\zeta_D}) = \sum_{(n_1, ..., n_D) \in \mathbb{N}^D} p(n_1, ..., n_D) \prod_{i=1}^{D} r_i^{n_i}$$

We can now define a generalized multidimensional generating function for $\underline{Z_n}$. We will note in the following the generating function of ζ_i distributed as P_i as

$$g_i(r_1, ...r_D) = \mathbb{E}_{P_i}(r_1^{\zeta_i(1)} ... r_D^{\zeta_i(D)}) = \sum_{(n_1, ..., n_D) \in \mathbb{N}^D} P_i(n_1, ..., n_D) \prod_{i=1}^{D} r_i^{n_i} \tag{5}$$

We define now the generalized multidimensional generating function for $\underline{Z_1}$.

$$G : [0, 1]^D \to (\mathbb{R}^+)^D \qquad (r_1, ...r_D) \mapsto (g_1(r_1, ...r_D), ..., g_D(r_1, ..., r_D)) \tag{6}$$

Notice that G here takes into account all distributions.

We note in the following, for clarity reasons, $\underline{r} = (r_1, ..., r_D)$.

The same way, we now define $g_{i,\textcircled{n}}$ the generating function of $(\underline{Z_n})$ given that the n-th parent of $(\underline{Z_n})$ - that is the root of the tree we are looking at at generation n is of type i. Let us call its distribution $P_{i,n}$.

$$g_{i,\textcircled{n}}(r_1, ...r_D) = \mathbb{E}_{P_{i,n}}(r_1^{Z_n(1)} ... r_D^{Z_n(D)}) = \sum_{(n_1, ..., n_D) \in \mathbb{N}^D} P_{i,n}(n_1, ..., n_D) \prod_{i=1}^{D} r_i^{n_i} \tag{7}$$

On the same way, we define $G_{\textcircled{n}}(\underline{r})$ the generalized multi-type generating function by

$$G_{\textcircled{n}}(\underline{r}) = (g_{1,\textcircled{n}}(\underline{r}), ..., g_{D,\textcircled{n}}(\underline{r})) \tag{8}$$

$G_{\textcircled{n}}$ stores the information of the tree for any type of root.

Let us now go into computations to derive an expression for $G_{\textcircled{n}}$.

We have by the definition of $g_{i,\textcircled{n}}$,

$$\mathbb{E}(\mathbb{E}(r_1^{Z_n(1)}...r_D^{Z_n(D)}|Z_{n-1},\mathcal{P},\text{root of type } i)=g_{i,\textcircled{n}}(\underline{r})$$

Let us call the event $\{Z_{n-1},\mathcal{P},\text{root of type } i\}=A_{n,i}$

and

$$g_{i,\textcircled{n}}(\underline{r})=\mathbb{E}(\mathbb{E}(\prod_{l=1}^{D}\prod_{i=1}^{D}\prod_{t=1}^{Z_{n-1}(i)}r_l^{\zeta_i^{(t)}(l)}|A_{n,i}))$$

thanks to the expression of $\underline{Z_n}$ in 4.

And

$$\mathbb{E}(\prod_{l=1}^{D}\prod_{i=1}^{D}\prod_{t=1}^{Z_{n-1}(i)}r_l^{\zeta_i^{(t)}(l)}|A_{n,i})=\prod_{i=1}^{D}\mathbb{E}(\prod_{l=1}^{D}\prod_{t=1}^{Z_{n-1}}r_l^{\zeta_i^{(t)}(l)}|A_{n,i})$$

.

However, the $(\zeta_i^{(t)})_t$ are independent and identically distributed. We can thus get the product out of the expectation and write

$$\mathbb{E}(\prod_{l=1}^{D}\prod_{i=1}^{D}\prod_{t=1}^{Z_{n-1}(i)}r_l^{\zeta_i^{(t)}(l)}|A_{n,i})=\prod_{i=1}^{D}\mathbb{E}(\prod_{l=1}^{D}r_l^{\zeta_i^{(t)}(l)}|A_{n,i})^{Z_{n-1}(i)}$$

.

We recognize then $\mathbb{E}(\prod_{l=1}^{D}r_l^{\zeta_i^{(t)}(l)}|A_{n,i})$ to be equal to $g_i(\underline{r})$.

Hence, we have

$$\mathbb{E}(\prod_{l=1}^{D}\prod_{i=1}^{D}\prod_{t=1}^{Z_{n-1}(i)}r_l^{\zeta_i^{(t)}(l)}|A_{n,i})=\prod_{i=1}^{D}g_i(\underline{r})^{Z_{n-1}(i)}$$

.

And in the end,

$$g_{i,\textcircled{n}}(\underline{r})=\mathbb{E}(\prod_{i=1}^{D}g_i(\underline{r})^{Z_{n-1}(i)})$$

.

That is, for $n \in \mathbb{N}^*$, with the convention $g_{i,\textcircled{0}} = Id \quad \forall i \in \{1, ..., D\}$,

$$g_{i,\textcircled{n}}(\underline{r}) = g_{i,\textcircled{n-1}}(g_1(\underline{r}), ..., g_D(\underline{r})) \tag{9}$$

For $G_{\textcircled{n}}$, it means then

$$G_{\textcircled{n}} = G_{\textcircled{n-1}} \circ G \tag{10}$$

where $\circ$ is the multivariate composition of functions (between functions of $\mathbb{R}^D$ to $\mathbb{R}^D$).

Now, the goal is, as before, to determine criteria for the convergence of the tree (*i.e.*, if it is extinct).

First, let us notice that again, the g_i are convex (and even strictly convex under reasonable assumptions as before). Moreover, they are positive and increasing along each coordinate for every $\underline{r} \in [0,1]^D$.

We also have, in virtue of the derivation of functions,

$$G'_{\textcircled{n}}(\underline{r}) = G'_{\textcircled{n-1}}(G(\underline{r})) \times G'(\underline{r})$$

with $G'_{\textcircled{n}}(\underline{r}) = (\frac{\partial g_{\textcircled{n},i}}{\partial r_j})_{i,j \in \{1,...,D\}}$.

We also have $G'(1, ..., 1) = G'(\mathbf{1}) = (\mathbb{E}_{P_i}(Z_j))_{i,j \in \{1,...,D\}}$, where we note in the following the vector $\mathbf{1} = \big(1, 1, ...1\big)$

As the g_i are all convex and increasing along each coordinate, we can say that over the set $[0,1]^D$, G' is necessarily maximized for any norm at $\mathbf{1}$. Considering the norm of G' might prove that G is a contraction mapping, in which case we would have a unique fixed point. Let us compute the operator norm of $G'(\mathbf{1})$, that is, $\sup_{u \in [0,1]^D} ||G'(\mathbf{1})u||$.

3.2.4 Fixed Point Discussion and Extinction Probability in the multitype setting

The first intuition leads us to compute this operator norm for different norms for u, that is $\|.\|_1$, $\|.\|_2$ and $\|.\|_\infty$, in order to draw conclusion on the existence of fixed points other than $\mathbf{1}$. We will present the results for $\|.\|_1$ and $\|.\|_\infty$ in the following.

We have $G'(1)u = (\sum_{k=1}^{D} \mathbb{E}_{P_i}(Z_k)u_k)_i$ As all $\mathbb{E}_{P_i}(Z_k)$ are positive, the u taken to maximize the quantity above must have all negative or all positive coordinates. we can thus suppose without loss of generality that they are all positive, and

$$\|G'(1)u\|_1 = \sum_{i=1}^{D}\sum_{k=1}^{D} \mathbb{E}_{P_i}(Z_k)u_k$$

We maximize this quantity with respect to u_k for each k, and in the end:

$$\frac{\partial \|G'(1)u\|_1}{\partial u_k} = \sum_{i=1}^{D} E_{P_i}(Z_k)$$

In order to maximize $\|G'(1)u\|_1$, under the constraint $\|u\|_1 = 1$, we must thus choose $u = (0, ..., 0, 1, 0, ..., 0) = e_{k_0}$ such that $k_0 = \arg\max_{1\leq k\leq D} \sum_{i=1}^{D} E_{P_i}(Z_k)$. We thus have

$$|||G'(1)|||_1 = \max_{1\leq k\leq D} \sum_{i=1}^{D} E_{P_i}(Z_k)$$

On the same way,

$$\|G'(1)u\|_\infty = \max_{1\leq i\leq D} \sum_{k=1}^{D} \mathbb{E}_{P_i}(Z_k)u_k$$

We can thus maximize this quantity by taking $u = \mathbf{1}$ and in that case,

$$|||G'(1)|||_\infty = \max_{1\leq i\leq D} \sum_{k=1}^{D} E_{P_i}(Z_k)$$

We recognized here the maximum expectation of the length of the rules for each B_i

To have G a contraction mapping, we should have $|||G'(1)||| < 1$. However, in the case of the infinity norm, $|||G'(1)|||_\infty < 1$ means that for all i, the expectation of the length of a rule must be strictly inferior to 1. However, we proved in the first part of this section that it almost surely did not happen.

Moreover, for the norm L1, this could be even larger as it is not bounded $|||G'(1)|||_1 = \max_{1\leq k\leq D} \sum_{i=1}^{D} E_{P_i}(Z_k)$ as D the number of types increases: for example, for each B_i, except B_1, we could have $E_{P_i}(Z_1) = 1$. Then $|||G'(1)|||_1$ would explode linearly in D without being necessarily unreasonable in terms of length: (for all $j \leq D$, $B_0 \to B_j$, $B_1 \to a$, and for $2 \leq i \leq D$, $B_i \to a'B_1$ forms a grammar with the properties described and that does not generated sentences longer than 2 words, but $|||G'(1)|||_1$ explodes in D).

Trying to find a contraction mapping criterion for G did not prove conclusive then, and we will try in the following to draw conclusions given fixed points for G.

Let us define now the sequence $(u_n)_\mathbb{N}$ of $\mathbb{R}^D$ recursively:

- $u_0 = (0, ..., 0)$
- $u_{n+1} = G(u_n)$

We have, by an immediate recurrence, that $u_n = G^{\textcircled{n}}(\underline{0})$

Hence, in regard of our previous points concerning the $g_{i,\textcircled{n}}$, we have that

$$u_n = P(Z_n(1) = ... = Z_n(D) = 0)$$

.

Moreover, as G is a bounded function of $[0,1]^D$ to $[0,1]^D$, we have that u_n is bounded. We thus have a sub-sequence that is convergent in $[0,1]^D$. We also have an increasing coordinate by coordinate as all g_i are increasing, and thus we have the convergence of the whole sequence u_n.

We could also have assessed the increasing of the sequence coordinate by coordinate by noticing that the event $Z_n(1) = ... = Z_n(D) = 0$ is included in the event $Z_{n+1}(1) = ... = Z_{n+1}(D) = 0$.

The limit must thus be a fixed point of G, by the theory of recursive sequences, and let us show that limit is the minimum coordinate by coordinate of all fixed points (which means in particular that if there are two fixed points, $\mathbf{1}$ is not the limit).

Let r_1^*, r_2^* be two fixed points of G. We will show by induction that each $g_{i,\textcircled{n}}(\underline{0})$ is inferior for all n to $(min(r_1^*(i), r_2^*(i)))$, for all $1 \leq i \leq D$. It is obviously true at rank 0. Suppose now it is true at rank n.

We have $G^{\textcircled{n+1}}(\underline{0}) = G^{\textcircled{n}} \circ G(\underline{0})$. Let j be in $1, ..., D$. Denote $a_j = g_j(g_{1,\textcircled{n}}(\underline{0}), ..., g_{D,\textcircled{n}}(\underline{0}))$ and we have each of the $g_{i,\textcircled{n}}(\underline{0}) \leq min(r_1^*(i), r_2^*(i))$. Suppose here without loss of generality that $min(r_1^*(j), r_2^*(j)) = r_1^*(j)$. We have $g_{i,\textcircled{n}}(\underline{0}) \leq r_1^*(i)$, and in particular, as g_j is increasing on each direction, we have $a_j \leq g_j(r_1^*(1), ..., r_1^*(D))$.

Hence $a_j \leq r_1^*(j)$ as r_1^* is a fixed point of G. Our induction is thus complete. By an immediate generalization, we can extend to results to the minimum coordinate by coordinate to all fixed points of G. Call that minimum by coordinate vector r^*. We know that u_n converges towards a fixed point of G, called r_u for example. Then, coordinate by coordinate, we have $r_u \geq r^*$ by definition of r^*, and by the recurrence, for each n, $u_n \leq r^* \leq r_u$.

Hence, we have $r_u = r^*$ and u_n converges toward the fixed point r^* with minimal coordinates among all other fixed points, and, if the type of the root is j, the probability of extinction of the tree is $r^*(j)$.

We thus have extended the Galton Watson extinction theory to multi-type trees, and thus to our problem here.

4 Simulation Study

In the light of the theoretical outcomes presented before, we conducted a simulation study for different values of the different hyper-parameters of the problems for the prior distribution of our grammar.

We will first describe the code used before presenting the results obtained.

4.1 Presentation and Architecture of the code

We will use Python object oriented abilities to construct different classes for modelling the hierarchy of our model here.

First, we construct the object `DP_B`, the Chinese Restaurant Process on the latent variables (the $(B_i)_{i \in \mathbb{N}}$). This was noted as $\mathbb{Q}$ in the formal presentation of our model. It will have three attributes: `alpha`the dispersion parameter for the process, `tables`a list representing all the tables discovered so far, stocking the number of customers sitting down at each table, and `n`the total number of customers so far. We implement as well two methods in that class: `__init__`for initialization of the Chinese Restaurant Process, which only takes `alpha`in argument, and creates a Chinese Restaurant Process with dispersion parameter `alpha,`and one table with one customer on it. We also create the method `new_B`which models the arrival of a new customer in the restaurant on the standard Chinese restaurant process procedure. It returns afterwards the table chosen (it samples from the current Chinese Restaurant Process).

Second, we construct `DP_rules`(respectively `DP_rules_reg`), the Chinese Restaurant process associated with one of the latent variables B_i describing the rules in the context free case (respectively in the regular case. This begins to be more complicated as the hierarchy is starting to appear. This was noted as P in the formal presentation of our model. The attributes will here be:

- again, `alpha1`, the dispersion parameter for the Dirichlet Processes on the rules.
- The `index` i of the latent variable associated with the Dirichlet Process we are constructing
- `rules` a list of the rules themselves (Reminder: a rule will be of the form $B_i \rightarrow a_j B_{j_1}...B_{j_k}$) so here rule will be modelled here as a sequence of latent variables (stocked here as $[j_1, ..., j_k]$. Those rules are the tables of our Chinese Restaurant Process.
- `count` an array associated with the list `rules,`counting the number of customers at each table. Thus, `count` and `rules` will be of the same length at all time.
- `ni` the number of customers for the Dirichlet process associated with the i-th latent variable B_i.

- lambd the parameter for the Poisson law regulating the length of each rule, or respectivelypthe parameter for the Bernoulli law in the regular case.

As before, we build a few functions for the evolution of this Dirichlet Process.

- First, a function __init__ initializing the Dirichlet Process with all its attributes.
- Second, new_rule adds a table to the Chinese Restaurant Process. The function is here separated from the global evolution function for clarity. It takes also in argument DP_Ba Dirichlet process on the latent variables themselves. We first generate a Poisson variable (respectively a Bernoulli variable)len of parameter lambd(respectively p), then sample len latent variables from DP_B and stack them to create our rule. We then add this rule to the list rules, and update counts accordingly.
- Finally, the function rulesamples from the Chinese Restaurant Process on a more classic way.

Finally, the class Sentence(respectively, Sentence_reg) describes the grammar sampling sentences (or, more accurately, their length only as we are not interested in the words themselves here). The attributes are again more complicated to model the hierarchy between the different Dirichlet processes.

- The length len of the sentence.
- The stack of variables stackmodelling the latent variables still to apply.
- The Dirichlet Process on the latent variables DPB of the class DP_B presented above.
- A list of Dirichlet Processes on rules of class DP_rules(respectively DP_rules_reg) associated with a latent variables DP_i which will evolve as variables appear.
- An array appearedwhich remembers which latent variable has appeared in the rules and thus has a Dirichlet Process associated in DP_i.
- A parameter for the Poisson law for rules lambd,or respectively the Bernoulli parameter p.
- A dispersion parameter alpha for the Dirichlet Process on the latent variables DPB.
- A dispersion parameter alpha1 for the Dirichlet Processes on the rules associated with one latent variable DP_i.

The class Sentence(or Sentence_reg) has two functions. One for initialization as always, __init__, which takes all the attributes as arguments, except for len which is initialized to 0 and stackwhich is initialized to the array [1] (we only have the start symbol at first in the stack). It is important to notice that here DPB the Dirichlet process on the $(B_i)_{i \in \mathbb{N}}$, DP_i the list of Dirichlet processes on the rules for each B_i and appeared are given as arguments, which allow us to generate multiple sentences associated with the same grammar (Again, the grammar here is entirely defined by DPB the non-terminal symbols and DP_i the set of rules associated).

The other function is `next_step`, which is applied directly on the sentence without any further arguments. It applies the first variable in the stack and upload then the stack and the length of the sentence. If the variable has appeared, it generates a rule from the Dirichlet Process associated and uploads the stack. Else, it creates the Dirichlet Process associated with that variables, generates a rule from that Dirichlet Process, and uploads `DP_i`, `appeared`, `stack`, and `len`. Moreover, it uses the function of the classes `DP_B` and `DP_rules` to generates rules and variables, which means it modifies consequently the attributes of the sentence as new variables and rules are sampled. The trick is also that it returns a boolean depending on if the stack is empty or not, or if the sentence is said to be infinite (Here, we will consider the sentence infinite when its length is over 100 - we could do more, but for computation efficiency, 100 is a good choice - especially as if not said to be infinite, the sentence length does not normally goes over 20). That way, we know if the sentence is over or not and when to switch to the next.

In the end, to simulate multiple sentences for a grammar, we initialize and object of the class `Sentence` and generate multiple sentences until using the same parameters until they are over. Reusing the previous `DP_B`, `DP_rules` and `appeared` makes sure we are using the same grammar.

In order to gain information as well on the variable $\mathbb{E}(L|P, \mathbb{Q}, \text{sentence finite})$ where $P, \mathbb{Q}$ are defined in the Presentation of the linguistics grammar modelling, we generate multiple grammars (`DP_B` and `DP_rules`in our code) from which we generate multiple sentences. We can thus build an empirical distribution of $\mathbb{E}(L|P, \mathbb{Q}, \text{sentence finite})$, and then vary hyper-parameters to get some intuition about their role. As well as computing $\mathbb{E}(L|P, \mathbb{Q}, \text{sentence finite})$, we can also compute the median of the length of sentences. This method is more robust as well, as it does not need to truncate the distribution at 100, as long as the median does not overpass 100, and which will be directly comparable to the results in the regular case (where all sentences are finite almost surely). We can also compute, in the context free model, a probability of extinction that will vary according to our hyperparameters, and help us get some intuition as well.

4.2 Simulation results

On the following figures, we present the results of the simulation of the length of sentences generated from different grammars, with different sets of hyperparameters. The error bars on Figure 2 represent a 95% Confidence Interval for $\mathbb{E}(L|P, \mathbb{Q}, \text{sentence finite})$, computed with their empirical distribution (which is approximated using a large number of Dirichlet processes - $\mathbb{Q}, P$ - , computing $\mathbb{E}(L|P, \mathbb{Q}, \text{sentence finite})$ for each. We kept on purpose just the largest on top, to show that the approximation of the event $\{\text{Sequence infinite}\}$ by the event $\{\text{length of the sentence} \leq 100\}$ is very reasonable.

On both Figure 2 and Figure 3, we can see a general trend. Not only does the length expectation decrease with the λ parameter in the Poisson law, as expected, but we also have a distinct decrease of the length expectation or median both on the M_1 concentration parameter of the Dirichlet Process on the non terminal symbols B_i, but also on the M_2 concentration parameter of the Dirichlet Process on the rules, generated from each B_i.

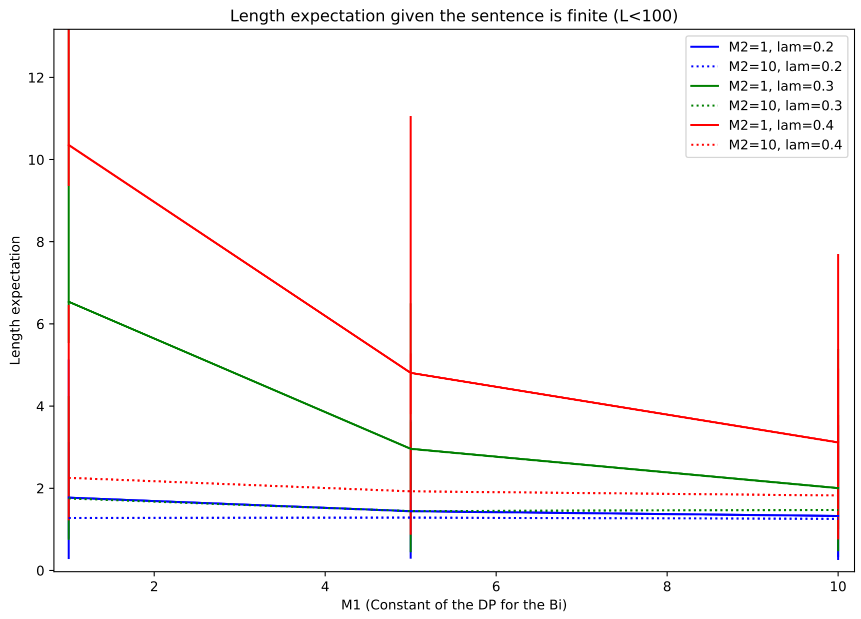

Figure 2: Expectation of the length of a sentence generated by a Context Free Grammar

We can interpret that decrease the following way. The higher the concentration parameter, the flatter and well distributed along the base distribution the Dirichlet Process will be. In that case, it means the higher the concentration parameter, the closer the distribution of the length of the rules will be to the original Poisson Law. Hence, the higher the concentration parameter, the more controllable the length of the sequences.

Finally, it appears that the parameter M_2 has a stronger effect on that decrease than M_1, which is reasonable according to the same reasoning, considering the fact that it controls more closely the distribution of the length of the rules for each variable B_i.

The behaviour of the probability of extinction is quite intuitive, once we described those behaviours. Indeed, a better control of the distribution of length drawn from the Dirichlet Processes induces an increase in the probability of extinction. However, we can argue that this change in the probability of extinction is really small (around 10^{-4}) so all parameters are quite relevant here. One might then argue that studying higher values of λ or smaller values for M_1 and M_2 could be also interesting, but it did not prove however conclusive regarding the length of the sentences produced.

Let us now consider the sentences generated from regular grammars, to see if any differences can be spotted regarding the length in our prior models, with the close sets of hyperparameters.

It appears the sentences produced are of the same length and that $\mathbb{E}_{H_0}(L|P,\mathbb{Q})$ seems to be distributed quite the same way. It can be considered logical, considering the fact that using a Poisson law with parameter λ in $[0.2, 0.5]$ puts the 90 to 98% of the probability mass between 0 and 1.

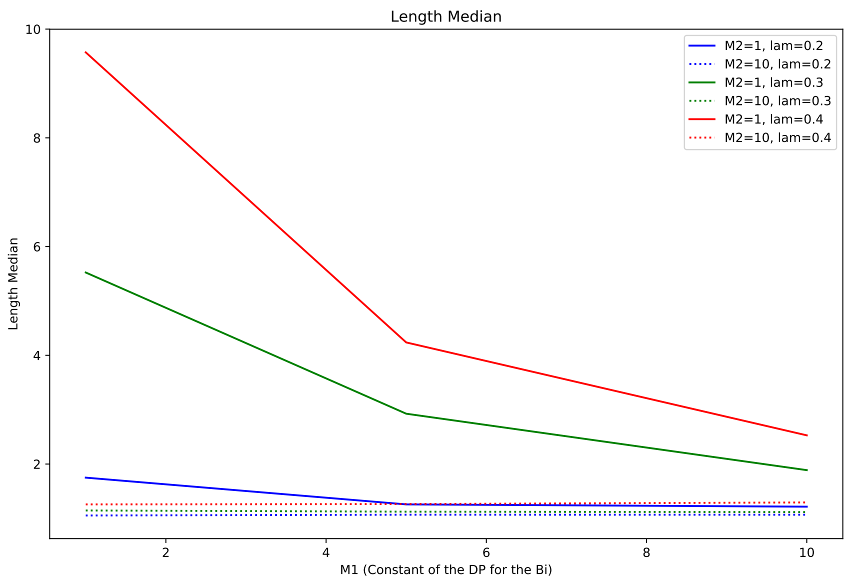

Figure 3: Median of the length of a sentence generated by a Context Free Grammar

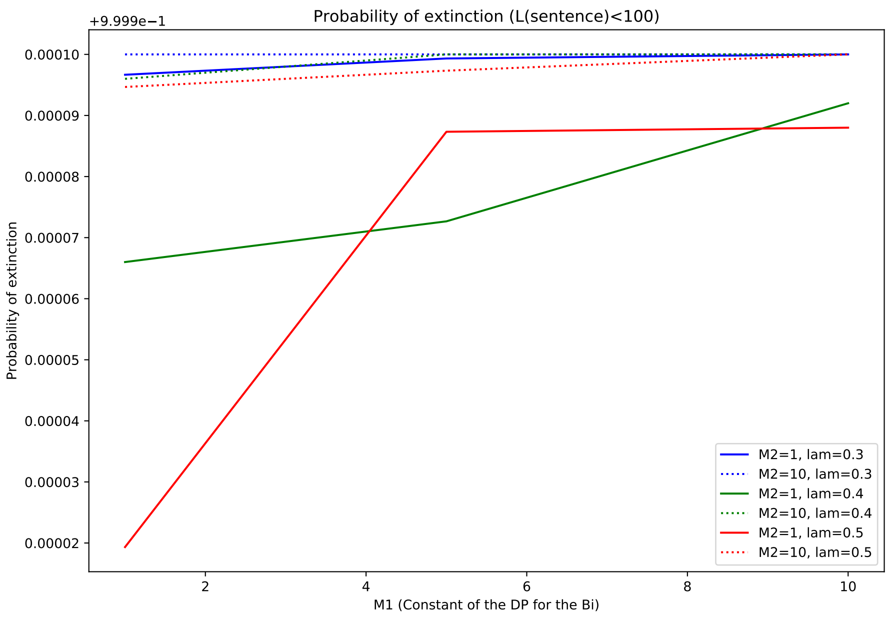

Figure 4: Probability of convergence of a sentence generated by a Context Free Grammar

In the end, our simulations showed how the length of the sentences and their convergence could be functions of the hyperparameters, and we determined a global set of hyperparameters that could be relevant for our study, typically, the hyperparameters presented here (all other tries for hyperparameters were not represented for reasons of clarity and lightness).

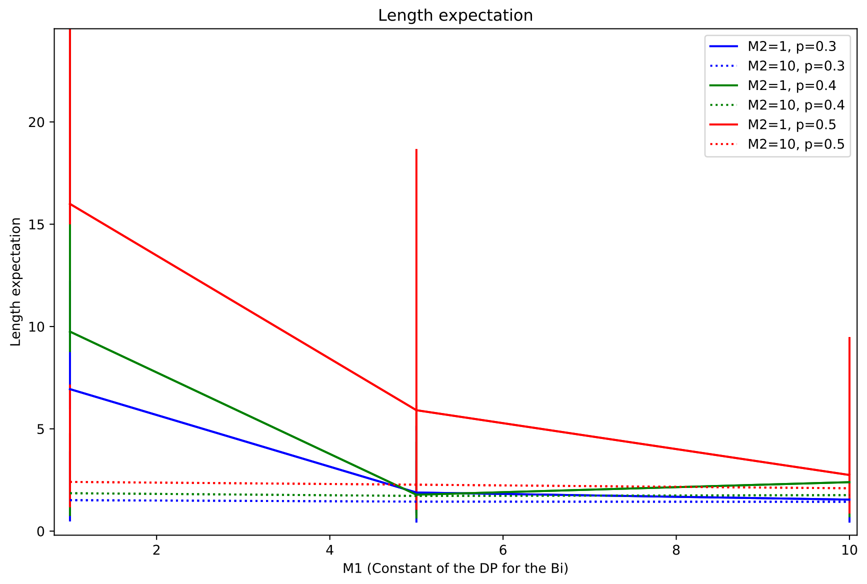

Figure 5: Expectation of the length of a sentence generated by a Regular Grammar

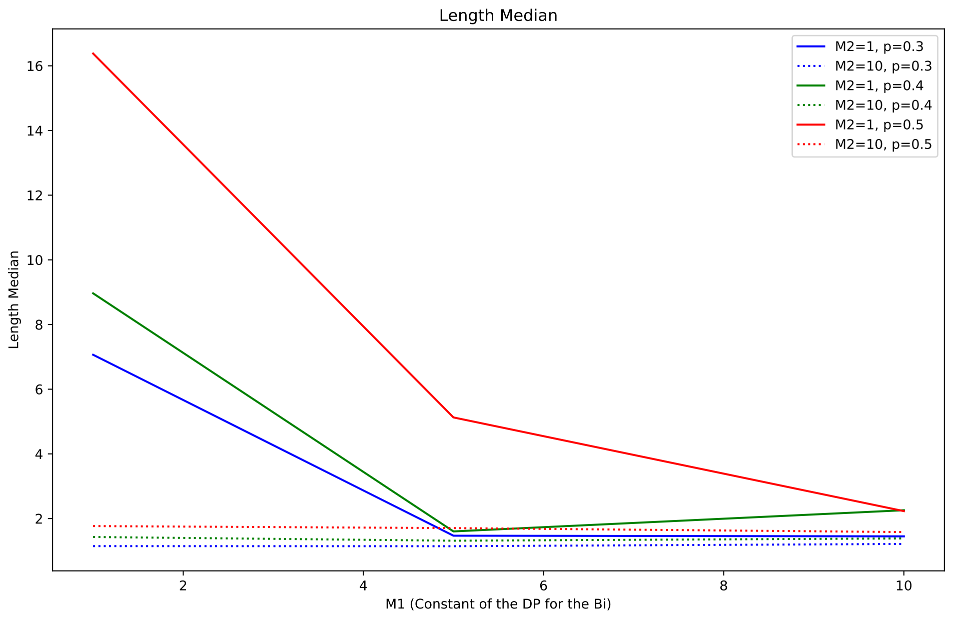

Figure 6: Median of the length of a sentence generated by a Regular Grammar

Moreover, the simulations showed our two prior models (regular and context free) are quite accurate: they could represent sentences of the same length, which means the only thing that would increase Bayes Factor in one way or the other would be the actual complexity of the sentences, and not their length.

5 Conclusion

This dissertation presented the baseline for a model to assess the complexity of a formal grammar, discerning the regular and context free classes for complexity. The model presented existed already and was well defined, however the method and the results produced were not that evident. In this work (and in the paper to come), we presented a proof that the Bayes Factor to compare marginal likelihoods in order to draw a line between regular and context free grammars was well defined, which did not come off that easily considering the various forms the rules can be shaped into, making the border blurry between regular and context free grammars in the generalized case. Moreover, we extended the work on the probability of extinction of Galton Watson trees on (finite) multi-typed trees, deriving a generalized generating function and extinction probability. However, it is still not clear what happens in the infinite multi type case, or when D the number of types goes to infinity. A continuation of the work here would then be to use the Sethuraman form of the Dirichlet Process to derive bounds or approximation when D goes to infinity, or determine the behaviour of the fixed point of G the generalized generating function when D the number of types goes to infinity. This work will be continued in prevision of the paper to come with Lawrence Murray, Robin Ryder and Judith Rousseau. Finally, all work has here been done on the prior model, to give us intuition or theoretical beliefs. It would be a natural extension of the work as well to see to what extent the posterior will differ in terms of length expectation or probability of extinction of the tree.

6 Bibliography

References

Abraham, R. and Delmas, J.-F. (2015). An introduction to galton-watson trees and their local limits. *arXiv preprint arXiv:1506.05571*.

Abraham, R. and Delmas, J.-F. (2016). Critical multi-type galton-watson trees conditioned to be large. *arXiv preprint arXiv:1511.01721v2*.

Blackwell, D. and MacQueen, J. B. (1973). Ferguson distributions via polya urn schemes. *Ann. Statist.*, 1(2):353–355.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.

D.J., A. (1985). Exchangeability and related topics. *Lecture Notes in Mathematics, Springer, Berlin, Heidelberg*, 1117.

Ferguson, T. S. (1973). A bayesian analysis of some nonparametric problems. *Ann. Statist.*, 1(2):209–230.

Greibach, S. A. (1965). A new normal-form theorem for context-free phrase structure grammars. *J. ACM*, 12(1):42–52.

Jager, 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.

Jiang X, Long T, C. W. L. J. D. S. W. L. (2018). Production of supra-regular spatial sequences by macaque monkeys. *Current Biology*, 18.

Ouattara, K., Lemasson, A., and Zuberbühler, K. (2009). Campbell's monkeys concatenate vocalizations into context-specific call sequences. *Proceedings of the National Academy of Sciences*, 106(51):22026–22031.

Schlenker, P., Chemla, E., Arnold, K., Lemasson, A., Ouattara, K., Keenan, S., Stephan, C., Ryder, R., and Zuberbühler, K. (2014). Monkey semantics: two âĂŸdialects' of Campbell's monkey alarm calls. *Linguistics and Philosophy*, 37(6):439–501.

Seneta, E. (1970). Population growth and the multi-type galtonâĂŞwatson process. *Nature*, 2012.

Sethuraman, J. (1994). A constructive definition of dirichlet priors. *Statistica Sinica 4*, pages 639–650.

Watson, H. W. and Galton, F. (1875). On the probability of the extinction of families. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 4:138–144.

Yee Whye Teh, Michael I. Jordan, M. J. B. and M.Blei, D. (2016). Hierarchical dirichlet processes. *arXiv preprint arXiv:1511.01721v2*.

7 Appendix

7.1 Python code

```
import numpy as np
import matplotlib.pyplot as plt


class DP_B:
    "Definition␣of␣the␣DP␣on␣the␣latent␣B␣variables"

    def __init__(self, alpha):
        self.alpha = alpha
        self.tables = np.ones(1)
        self.n = 1

    def new_B(self):
        rand = np.random.uniform(low= 0., high = 1., size = 1)
        lb = 0
        for i in range(self.tables.size):
            if rand < (lb + self.tables[i]/(self.n + self.alpha)):
                self.tables[i] += 1
                self.n +=1
                return(i)
                break
            lb += self.tables[i]/(self.n + self.alpha)
    ##New table created
        self.tables = np.append(self.tables, 1)
        return(self.tables.size)


######################################################################
#################CONTEXT FREE GRAMMARS################################
######################################################################
class DP_rules:
    "Definition␣of␣the␣DP␣on␣the␣rules␣for␣each␣B_i"

    def __init__(self, alpha1, i, lambd):
        self.alpha1 = alpha1
        self.rules = list()
        self.count = np.ones(0)
        self.index = i
        self.ni = 0
        self.lambd = lambd
    ##Definition of a new rule (sequence of numbers
    ##for the latent variables)
    def new_rule(self, DPB):
        len = np.random.poisson(lam = self.lambd)
        #print(len)
        r = np.ones(0)
        if (len==0):
            return(r)
        for k in range(len):
            ##Actualize the DP associated with all variables
            r = np.append(r, DPB.new_B())

        return(r)




    def rule(self, DPB):
        rand = np.random.uniform(low= 0., high = 1., size = 1)
        ##Initialization: create a new rule
        if (self.ni==0):
            self.rules.append(self.new_rule(DPB))
            self.ni = 1
            self.count = np.ones(1)
            return(self.rules[0])
        else:
            rand = np.random.uniform(low= 0., high = 1.,

            lb = 0
            for i in range(len(self.rules)):
                if rand<(lb+self.count[i]/(self.ni+self.alpha1)):
                    ##Increment the counting for specified rule
                    self.count[i] +=1
                    self.ni +=1
                    return(self.rules[i])
                    break
                lb += self.count[i]/(self.ni + self.alpha1)
            r = self.new_rule(DPB)
            self.rules.append(r)
            self.count = np.append(self.count, 1)
            return(r)



class Sentence:
    "Definition␣of␣a␣grammar␣producing"
    "a␣sentence␣-␣without␣words,␣just␣its␣construction"

    def __init__(self, DPB, DP_i, lambd, alpha, alpha1, appeared):
        self.len = 0
        self.stack = np.ones(1)
        self.DPB = DPB
        self.DP_i = DP_i ##list of DP for the rules
        self.lambd = lambd
        self.alpha = alpha
        self.alpha1 = alpha1
        self.appeared = appeared




    def next_step(self):
        if self.stack.size == 0:
            #print('Sentence over')
            return(False)
        if self.len == 200:
            print('Sentence␣infinite')
            return(False)
        else:
            ##If the variable has appeared
            if self.stack[0] in self.appeared:
                current = self.stack[0]
                index = self.appeared.index(current)
                ##Take the DP associated with that variable
                ##and generate a rule
                aux = self.DP_i[index].rule(self.DPB)
                ##Actualize the stack
                self.stack = np.insert(self.stack[1:], 0, aux)
                ##increment the length of the sentence
                self.len += 1
            else:
                current = self.stack[0]
                ##Actualize the appeared list of
                ##already seen variables
                self.appeared.append(current)
                ##Actualize the list of DP of rules accordingly
                self.DP_i.append(DP_rules(alpha1 = self.alpha1,
                                          i = current,
                                          lambd = self.lambd))

                ##Actualize the stack and the length
                index = self.appeared.index(current)
                aux = self.DP_i[index].rule(self.DPB)
                self.stack = np.insert(self.stack[1:], 0, aux)
                self.len += 1
        return(True)



################################################################################
######################REGULAR GRAMMARS##########################################
################################################################################


class DP_rules_Reg:
    "Definition␣of␣the␣DP␣on␣the␣rules␣for␣each␣B_i"

    def __init__(self, alpha1, i, p):
        self.alpha1 = alpha1
        self.rules = list()
        self.count = np.ones(0)
        self.index = i
        self.ni = 0
        self.p = p


    ##Definition of a new rule (sequence of numbers for the
    ##latent variables)
    def new_rule(self, DPB):
        len = np.random.binomial(n=1, p = self.p)
        #print(len)
        r = np.ones(0)
        if (len==0):
            return(r)
        for k in range(len):
            ##Actualize the DP associated with all variables
            r = np.append(r, DPB.new_B())

        return(r)




    def rule(self, DPB):
        rand = np.random.uniform(low= 0., high = 1., size = 1)
        ##Initialization: create a new rule
        if (self.ni==0):
            self.rules.append(self.new_rule(DPB))
            self.ni = 1
            self.count = np.ones(1)
            return(self.rules[0])
        else:
            rand = np.random.uniform(low= 0., high = 1.,

            lb = 0
            for i in range(len(self.rules)):
                if rand<(lb+self.count[i]/(self.ni+self.alpha1)):
                    ##Increment the counting for the specified rule
                    self.count[i] +=1
                    self.ni +=1
                    return(self.rules[i])
                    break
                lb += self.count[i]/(self.ni + self.alpha1)
            r = self.new_rule(DPB)
            self.rules.append(r)
            self.count = np.append(self.count, 1)
            return(r)




class Sentence_reg:
    "Definition␣of␣a␣grammar␣producing␣a"
    "sentence␣–␣without␣words,␣just␣its␣construction"

    def __init__(self, DPB, DP_i, p, alpha, alpha1, appeared):
        self.len = 0
        self.stack = np.ones(1)
        self.DPB = DPB
        self.DP_i = DP_i ##list of DP for the rules
        self.p = p
        self.alpha = alpha
        self.alpha1 = alpha1
        self.appeared = appeared





    def next_step(self):
        if self.stack.size == 0:
            #print('Sentence over')
            return(False)
        if self.len == 200:
            print('Sentence␣infinite')
            return(False)
        else:
            ##If the variable has appeared
            if self.stack[0] in self.appeared:
                current = self.stack[0]
                index = self.appeared.index(current)
                ##Take the DP associated with that
                ##variable and generate a rule
                aux = self.DP_i[index].rule(self.DPB)
                ##Actualize the stack
                self.stack = np.insert(self.stack[1:], 0, aux)
                ##increment the length of the sentence
                self.len += 1
            else:
                current = self.stack[0]
                ##Actualize the appeared list of
                ##already seen variables
                self.appeared.append(current)
                ##Actualize the list of DP of rules accordingly
                self.DP_i.append(DP_rules_Reg(alpha1 = self.alpha1,
                                              i = current,
                                              p = self.p))

                ##Actualize the stack and the length
                index = self.appeared.index(current)
                aux = self.DP_i[index].rule(self.DPB)
                self.stack = np.insert(self.stack[1:], 0, aux)
                self.len += 1
        return(True)
```

```
###Simulation Part

alpha0vec = [1,5,10]

alpha1vec = [1,5,10]
```

```
lambvec = np.linspace(start = 0.2, stop = 0.4, num = 3)


m_res = np.zeros((3,3,3))
qm95 = np.zeros((3,3,3))
qm5 = np.zeros((3,3,3))
sd_res = np.zeros((3,3,3))
qsd95 = np.zeros((3,3,3))
qsd5 = np.zeros((3,3,3))
p_res = np.zeros((3,3,3))
med_res = np.zeros((3,3,3))
N_dp = 500


for i1 in range(3):
    for i2 in range(3):
        for i3 in range(3):
            maux = list()
            sdaux = list()
            medaux = list()
            paux = list()
            for l in range(N_dp):
                a = alpha0vec[i1]
                a1 = alpha0vec[i2]
                lam = lambvec[i3]



                DPB99 = DP_B(alpha = a)
                sentence = Sentence(DPB= DPB99,
                                    DP_i = list(),
                                    lambd = lam,
                                    alpha = a,
                                    alpha1= a1,
                                    appeared = list())

                i=0
                while (sentence.next_step()):
                    i+=1

                N = 3000
                lengths_res = list()
                lengths_all = list()
                for k in range(N):
                    sentence = Sentence(DPB= DPB99,
                                            DP_i = sentence.DP_i,
                                          lambd = lam,
                                            alpha = a,
                                            alpha1= a1,
                                      appeared = sentence.appeared)


                    i=0
                    while (sentence.next_step()):
                        i+=1
                    lengths_all.append(sentence.len)
                    if sentence.len!=100:
                        lengths_res.append(sentence.len)
                maux.append(np.mean(lengths_res))
                sdaux.append(np.std(lengths_res))
                medaux.append(np.median(lengths_all))
                paux.append(len(lengths_res)/N)

            p_res[i1,i2,i3] = np.mean(paux)
            m_res[i1,i2,i3] = np.mean(maux)
            qm95[i1,i2,i3] = np.quantile(maux, 0.975)
            qm5[i1,i2,i3] = np.quantile(maux, 0.025)
            sd_res[i1,i2,i3] = np.mean(sdaux)
            qsd95[i1,i2,i3] = np.quantile(sdaux, 0.975)
            qsd5[i1,i2,i3] = np.quantile(sdaux, 0.025)
            med_res[i1,i2,i3] = np.mean(medaux)



colors = ['b', 'g', 'r', 'c', 'm', 'y']
fm = plt.figure()
for i in range(3):
    plt.plot(alpha0vec, m_res[:, 0, i], c = colors[i])
    plt.errorbar(alpha0vec, m_res[:, 0, i],
                yerr = np.vstack((qm5[:,0,i],qm95[:,0,i])),
                 c = colors[i])
    plt.plot(alpha0vec, m_res[:, 2, i], c = colors[i], ls =':')
    plt.errorbar(alpha0vec, m_res[:, 2, i],
                yerr = np.vstack((qm5[:,2,i],qm95[:,2,i])),
                  c = colors[i], ls = ':')
```

```
plt.legend(('M2=1,␣lam=0.2', 'M2=10,␣lam=0.2',
            'M2=1,␣lam=0.3', 'M2=10,␣lam=0.3',
            'M2=1,␣lam=0.4',  'M2=10,␣lam=0.4'))
plt.title('Length␣expectation␣given␣the␣sentence␣is␣finite␣(L<100)')
plt.xlabel('M1␣(Constant␣of␣the␣DP␣for␣the␣Bi)')
plt.ylabel('Length␣expectation')

plt.show()



for i in range(3):
    plt.plot(alpha0vec, med_res[:, 0, i], c = colors[i])
    plt.plot(alpha0vec, med_res[:, 2, i], c = colors[i], ls =':')

plt.legend(('M2=1,␣lam=0.2', 'M2=10,␣lam=0.2',
            'M2=1,␣lam=0.3', 'M2=10,␣lam=0.3',
            'M2=1,␣lam=0.4',  'M2=10,␣lam=0.4'))
plt.title('Length␣Median')
plt.xlabel('M1␣(Constant␣of␣the␣DP␣for␣the␣Bi)')
plt.ylabel('Length␣Median')

plt.show()



for i in range(3):
    plt.plot(alpha0vec, p_res[:, 0, i], c = colors[i])
    plt.plot(alpha0vec, p_res[:, 2, i], c = colors[i], ls =':')

plt.legend(('M2=1,␣lam=0.2', 'M2=10,␣lam=0.2',
            'M2=1,␣lam=0.3', 'M2=10,␣lam=0.3',
            'M2=1,␣lam=0.4',  'M2=10,␣lam=0.4'))
plt.title('Probability␣of␣extinction␣(L(sentence)<100)')
plt.xlabel('M1␣(Constant␣of␣the␣DP␣for␣the␣Bi)')
plt.ylabel('Probability␣of␣extinction')

plt.show()
```

```
###Simulation Part

alpha0vec = [1,5,10]

alpha1vec = [1,5,10]

lambvec = np.linspace(start = 0.3, stop = 0.5, num = 3)


m_res = np.zeros((3,3,3))
qm95 = np.zeros((3,3,3))
qm5 = np.zeros((3,3,3))
sd_res = np.zeros((3,3,3))
qsd95 = np.zeros((3,3,3))
qsd5 = np.zeros((3,3,3))
p_res = np.zeros((3,3,3))
med_res = np.zeros((3,3,3))
N_dp = 200



for i1 in range(3):
    for i2 in range(3):
        for i3 in range(3):
            maux = list()
            sdaux = list()
            medaux = list()
            for l in range(N_dp):
                a = alpha0vec[i1]
                a1 = alpha0vec[i2]
                lam = lambvec[i3]


                DPB99 = DP_B(alpha = a)
                sentence = Sentence_reg(DPB= DPB99,
                                        DP_i = list(),
                                        p = lam,
                                        alpha = a,
                                        alpha1= a1,
                                        appeared = list())

                i=0
    while (sentence.next_step()):
        i+=1

    N = 1000
    lengths_res = list()
    lengths_all = list()
    for k in range(N):
        sentence = Sentence_reg(DPB= sentence.DPB,
                                DP_i = sentence.DP_i,
                                p = lam,
                                alpha = a,
                                alpha1= a1,
                            appeared = sentence.appeared)
        i=0
        while (sentence.next_step()):
            i+=1
        lengths_all.append(sentence.len)
        if sentence.len!=100:
            lengths_res.append(sentence.len)
    maux.append(np.mean(lengths_res))
    sdaux.append(np.std(lengths_res))
    medaux.append(np.median(lengths_all))


m_res[i1,i2,i3] = np.mean(maux)
qm95[i1,i2,i3] = np.quantile(maux, 0.975)
qm5[i1,i2,i3] = np.quantile(maux, 0.025)
sd_res[i1,i2,i3] = np.mean(sdaux)
qsd95[i1,i2,i3] = np.quantile(sdaux, 0.975)
qsd5[i1,i2,i3] = np.quantile(sdaux, 0.025)
med_res[i1,i2,i3] = np.mean(medaux)
```

```
colors = ['b', 'g', 'r', 'c', 'm', 'y']
fm = plt.figure()
for i in range(3):
    plt.plot(alpha0vec, m_res[:, 0, i], c = colors[i])
    plt.errorbar(alpha0vec, m_res[:, 0, i],
                 yerr = np.vstack((qm5[:,0,i],qm95[:,0,i])),
                 c = colors[i])
    plt.plot(alpha0vec, m_res[:, 2, i], c = colors[i], ls =':')
    plt.errorbar(alpha0vec, m_res[:, 2, i],
                 yerr = np.vstack((qm5[:,2,i],qm95[:,2,i])),
                 c = colors[i], ls = ':')

plt.legend(('M2=1,␣p=0.3', 'M2=10,␣p=0.3',
            'M2=1,␣p=0.4', 'M2=10,␣p=0.4',
            'M2=1,␣p=0.5', 'M2=10,␣p=0.5'))
plt.title('Length␣expectation')
plt.xlabel('M1␣(Constant␣of␣the␣DP␣for␣the␣Bi)')
plt.ylabel('Length␣expectation')

plt.show()



for i in range(3):
    plt.plot(alpha0vec, med_res[:, 0, i], c = colors[i])
    plt.plot(alpha0vec, med_res[:, 2, i], c = colors[i], ls =':')

plt.legend(('M2=1,␣p=0.3', 'M2=10,␣p=0.3',
            'M2=1,␣p=0.4', 'M2=10,␣p=0.4',
            'M2=1,␣p=0.5', 'M2=10,␣p=0.5'))
plt.title('Length␣Median')
plt.xlabel('M1␣(Constant␣of␣the␣DP␣for␣the␣Bi)')
plt.ylabel('Length␣Median')

plt.show()
```