In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature _____

Date _____

Imposing Structure on Generated Sequences: Constrained Hidden Markov Processes

by

Porter W. Glines

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in the Department of Computer Science

Idaho State University

May 2022

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of Porter W. Glines find it satisfactory and recommend that it be accepted.

Paul M. Bodily,
Major Advisor

Leslie Kerby,
Committee Member

Irene van Woerden,
Graduate Faculty Representative

Table of Contents

List of Figures

List of Tables

Imposing Structure on Generated Sequences: Constrained Hidden Markov Processes

Thesis Abstract – Idaho State University (2022)

Markov models and neural networks are widely used in systems tasked with generating natural and meaningful sequences. Generating high-quality sequences often requires the system to impose structure on the sequences via user-defined control constraints. These models are typically not compatible with control constraints. Work has been done to combine non-hidden Markov models with constraints; however, this approach has the problem of diminishing solution space sizes for increasing Markov orders or constraint complexity. For neural networks, the anticipation-Recurrent Neural Network (anticipation-RNN) allows control constraints but is limited in what kind of constraints work effectively.

We propose an efficient method to apply control constraints to a hidden Markov model that, like the non-hidden variant, 1) guarantees sequences generated satisfy constraints and 2) the statistical distribution of the constrained model is the same as the original model. The proposed model satisfies the control constraint requirement and avoids the problem of diminishing solution space sizes afforded by the abstraction introduced by the hidden states.

Keywords: Sequence Generation, Markov Models, Constraint Satisfaction, Constrained Hidden Markov Process, Anticipation-RNN

Chapter 1

Introduction

Sequential content generation is a common task for A.I. systems. Systems generate musical sequences for interactive song creation [41], natural language sentences for counselor chatbots [39], narrative content for storytelling [50], etc. These systems are found in the context of consumer software, academic projects, and computational creativity. Conceptualizations used in these systems range from Markovian models [41] to Deep Learning models [24]. This thesis presents a novel Markovian model: a Constrained Hidden Markov Process (CHiMP). This new model will be compared against a related Markovian model and a neural network based model to demonstrate its abilities.

Digital media is ubiquitous. Personalized content generation systems have the potential to help people create meaningful content for themselves and for others. Creative content generation systems are being investigated in a therapeutic context where patients can gain therapeutic value out of expressing themselves with the help of a creative system [9]. These systems can be useful for those looking to generate non-copyrighted music (e.g., video makers) or plagiarism free text to fill articles [44].

While digital media comes in many forms, this thesis focuses on sequential content such as music and natural language text. Sequential content often has domain-specific structure that is important to the perceived cohesiveness or quality of a generated sequence. For example, in natural language sentences, the underlying subject-verb-object agreement is important to having a cohesive sentence [26]. In music, the presence of motifs can elevate a musical piece to be something more interesting [1, 38].

In some content generation systems, it is desirable for the model to be user-steerable such that a user can influence generated sequences by defining a set of constraints. A user might desire to steer a music generation system to follow a loosely "happy sounding" pitch contour or end musical phrases with a specific rhythm. User-steerable models are particularly desirable to co-creative systems where a creative A.I. system is supporting and building off of inputs of a human user.

The relation that structure has with cohesiveness and the desire for user-steerable models motivates the problem this thesis aims to address:

> *Problem: Imposing structure onto generated sequences where current models can adhere to constraints and maintain sequence cohesiveness, but have diminishing solution space sizes for restrictive constraints.*

When comparing solutions to this problem, it will be important to measure the trade-offs that imposing structure will have on model generalizing power and generated sequence cohesiveness. Model generalizing power can be greatly diminished when structure is imposed [21]; which is to say that as more constraints are applied to sequences, the smaller the solution space becomes. One way to combat diminishing solution space sizes is to introduce an element of abstraction into the model [21]. In our model, CHiMP, the element of abstraction is the hidden nodes in a hidden Markov process which are absent in a non-hidden Markov process. Introducing an element of abstraction into a model potentially introduces negative effects on the cohesiveness of generated sequences. However, we find that CHiMP does not generate sequences that are significantly less cohesive for the domain of music.

In comparing models that impose structure onto generated sequences, we measure how the imposition of structure effects the models generalizing power by measuring the percent of unique generated sequences from many samplings of the model and generated sequence cohesiveness through conducting an Institutional Review Board (IRB) approved survey.

The goal of this thesis is to evaluate how CHiMP compares against other sequence generation models in generating musical sequences that A) adhere to desired unary structural

constraints, B) maintain semantic cohesiveness, and C) have adequate expressive/generalizing power. This thesis also evaluates CHiMP in regards to its theoretical and experimental time complexities to train and generate sequences.

The remainder of this thesis, besides Chapter 2, is a compendium of published or soon to be published papers. Chapter 2 discusses related work and background areas. Chapter 3 presents a paper published at ICCC 2019 that applies the constrained Markov process (CoMP) to a generative problem and motivates the designing of CHiMP. Chapter 4 presents a paper presented at ICCC 2020 that motivates the need for models like CHiMP in the context of computational creativity. Chapter 5 presents a paper presented at i-ETC 2020 that initially describes CHiMP. Chapter 6 presents an article that fully describes CHiMP, including the methods for constructing the model, time complexity analysis, and experimental results. Finally, Chapter 6 discusses the overall contributions and conclusion.

Chapter 2

Related Work

2.1 Broad Research Area Overview

The concept of Markov chains has been known for over 100 years by mathematicians and engineers. Since the 1960's, optimizations and refinements afforded by algorithms such as the Baum-Welch method allowed Markov models to be used for more complex applications, namely speech processing and recognition [52]. Markov models are later used with success in biological sequence analysis where efficient computational models are needed to process large amounts of data [67]. While Markov models are known for their efficient pattern recognition capabilities, they can also be used for sequence generation. Markov models have been used successfully to generate music [41], natural language text [17], and speech synthesis [62]. Regardless of their application, Markov models are based on the "Markov property" which is that a future state is only dependent on the last state. Put in terms of the probabilities of a sequence $s_1, \ldots, s_n$, the Markov property states the following:

$$p(s_i|s_1, \ldots, s_{i-1}) = p(s_i|s_{i-1}).$$

In general, Markov models are trained by counting the number of occurrences and transitions between training words in a training dataset. Once the model is trained, sequences can be generated by performing a simple *random walk* on the model: an initial word is randomly chosen based on the count of word occurrences (called prior probabilities); then, the next word is randomly chosen based on the counted transitions from the initial word to

other possible words (called transition probabilities). The chosen next word is appended to the sequence and the process is iterated to produce a sequence. This process of sampling probabilities makes the Markov model stochastic.

As of the last few decades, a popular class of models has been neural networks [57]. A recurrent neural network (RNN) is a specialized neural network suited for sequence analysis as well as sequence generation [23, 46, 59]. Like Markov models, they have been used successfully in sequence based domains such as music [6, 19], natural language [61], and motion capture [60]. An RNN generates sequences by repeatedly predicting words in a sequence. For an RNN generating sequences, the network itself is deterministic; however, stochasticity is introduced when repeatedly sampling words and its predictive distribution depends on the previous words in the sequence [23]. In this way, RNNs store information about previous words in the sequence to make its predictions of future words better. However, RNNs are not able to store information about previous words further back in the sequence, i.e., a simple RNN has no long-term memory [29].

Long Short-term Memory (LSTM) is an RNN variant that is designed to maintain a longer term memory about previous words in the sequence [28]. LSTMs use multiple neural networks as well as memory cells to achieve this increased ability to remember past inputs. This allows LSTMs to display a better understanding of lasting structure and patterns in a sequence compared to simple RNNs.

Another variant of RNNs meant to replace the LSTM for some is the Transformer [63]. Unlike the LSTM, the Transformer does not need to process its data in a sequential computation. Instead, it relies on an attention mechanism which allows the Transformer to process data in parallel which significantly decreases training times. This allows the model to be able to train on huge datasets with a fraction of the training time usually needed. As such, the Transformer is seen as the current state of the art in natural language processing and is used in projects such as OpenAI's GPT-3 [8].

2.2 Constrained Models Research-Related Work

A common way of imposing structure onto generated sequences is to combine a model with constraints. The combination of models with constraints implies the model is then able to enforce user-defined constraints onto specific positions of a generated sequence. For example, a model might constrain a generated sequence to end a sentence with a word that rhymes with a word group. For this thesis, we will focus on unary constraints, i.e., constraints that only involve one sequence position. However, binary constraints for Markovian models are trivial to implement within the Markov scope.

Pachet et al. introduced the constrained non-hidden Markov process (CoMP) that is a successful model for generating constrained sequences [42]. The differentiating factor of CoMP and CHiMP is that CoMP uses a non-hidden Markov model and CHiMP uses a hidden Markov model. CoMP maintains the property of Markovian models in that it is simple to implement and efficient. CoMP has been used for music generation [42, 55], lyric generation [2], and avoiding plagiarism [44]. The problem with CoMP is that as more and more constraints are added, the solution space can diminish to the point of being unusable [21]. In other words, rigorous constraints can greatly diminish the generalizing ability of CoMP. Out of the models discussed in this section, CoMP is the most closely related model to CHiMP. However, where CoMP performs poorly under rigorous constraints, CHiMP performs well under many constraints due to its element of abstraction which is its hidden nodes.

A factor graph is a way of expressing a global function of several variables into a product of local functions [20]. Factor graphs allow for a unified way of modeling in topics such as signal processing, Kalman filtering, and Markov models [33]. In fact, factor graphs subsume Markov models as well as Bayesian networks which are a more generalized Markov model. The factor graph has been used to constrain a Markov model by composing a factor graph of binary factors that combine a Markov model with an automaton [45]. The automaton in the factors imposes unary constraints on the Markov model. This approach of using factor graphs to constrain Markov models is functionally equivalent to CoMP.

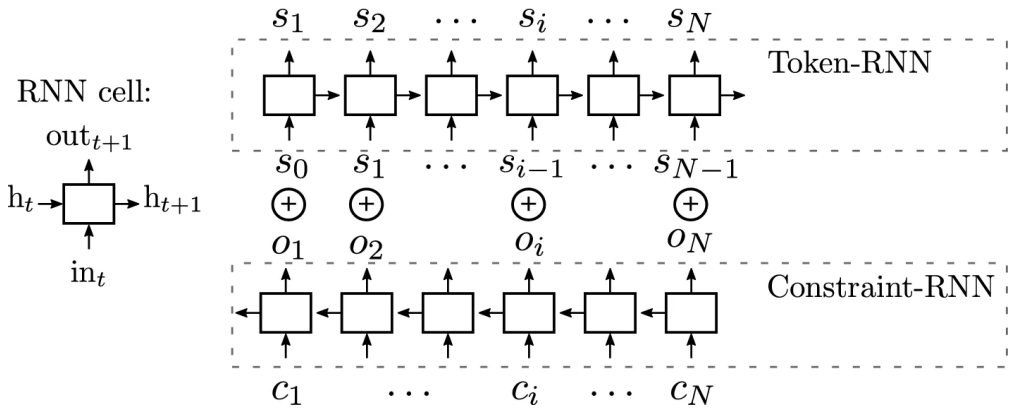


Figure 2.1: The anticipation-RNN architecture as depicted by Gaëtan Hadjeres and Frank Nielsen [24] where $s_1, \ldots, s_N$ sequence symbols are predicted given $c_1, \ldots, c_N$ constraint symbols and the previous sequence symbols $s_0, \ldots, s_{N-1}$. For each RNN cell, in_t is its input, out_{t+1} its output, and h_t its hidden state at time t.

Multi-valued decision diagrams (MDDs) are another way of expressing models and are common in the field of constraint programming. MDDs are a compressing data structure defined over a set of variables and used to store tuples of values [48]. MDDs, like factor graphs, have been used to combine Markov models with constraints. Perez et al. state that their use of MDDs was to demonstrate a more general way of constraining a Markov model than factor graphs.

A model commonly applied to the task of sequence generation is the RNN. RNNs, representing the neural network approach, are different from factor graphs, MDDs, and CoMP in that they perform higher-dimensional interpolation of training instances when making predictions [23]. This allows RNNs to exhibit more complex behavior through generated sequences. Due to the more complex behavior, RNNs are not affected by the zero-frequency problem as the probabilistic models are. However, RNNs typically do not make guarantees of satisfying user-defined constraints. An anticipation-RNN [24] aims to enforce unary constraints using a two stacked long short-term memory (LSTM) [23] architecture as shown in Figure 2.1. The anticipation-RNN is used in the DeepBach system and will be used for comparison with CHiMP [25]. The anticipation-RNN achieves success in constraining Bach chorales to user-defined unary constraints; however, the model does

allow a percentage of generated sequences that do not satisfy constraints and thus still does not make guarantees. For the anticipation-RNN, difficult or "out-of-style" constraints can even have a poor constraint satisfaction rate (less than 50% of generated sequences satisfy constraints) [24].

Chapter 3

"She Offered No Argument": Constrained Probabilistic Modeling for Mnemonic Device Generation

This paper was presented at ICCC 2019 and published in the conference proceedings pp. 81–88

3.1 Abstract

A common aspect to creativity as described by creative theorists is the juxtaposition and balance of two opposing qualities, namely novelty and typicality. Practical models of computational creativity are needed that effectively leverage the contributions of each of these qualities in a synchronous manner. We discuss the effectiveness of constrained probabilistic models in representing this duality in generative models of creativity. We illustrate constrained Markov models as an example of a constrained probabilistic model and demonstrate its application to computational creativity in the elaboration of a system called NhMMonic for generating mnemonic devices. We demonstrate the effectiveness of the system[1] using a qualitative survey. Our findings suggest that the constrained Markov model is particularly effective at generating mnemonics that exhibit novelty and typicality in grammatical and semantic flow with the overall result of more effective mnemonics for the purpose of memorization. Source code as well as our mnemonic device generator are both freely accessible online.

[1] An interactive demo can be viewed at `https://ccil.cs.isu.edu/projects/mnemonic/`

3.2 Introduction

Computational creativity (CC) has been defined as "the philosophy, science and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviours that unbiased observers would deem to be creative" [12]. The plural focus on the philosophy, science and engineering of computational systems has yielded valuable theoretical contributions as well as a number of functional creative systems. Emergent from this plural focus is the challenge of maintaining harmony between theory and practice. To be sure the abstract philosophy and concrete engineering can and should work to challenge one another in their mutual growth and evolution; however, the goal ultimately is to develop systems that accurately reflect the philosophical moorings and to advance theories whose tenets agree with what is observed about creativity in practice. Thus the role of *practical models* of creativity becomes significant—models that, by virtue of their ability to implement principles deriving from the philosophy, can be generalized beyond any single creative system with great effect, while maintaining ready applicability and implementability. As described by Jordanous [30], these models define the *creative process* of a system, namely "what the creative individual does to be creative."

Several examples of practical models of creativity have been demonstrated. Evolutionary models represent a practical implementation of the widely-accepted theory that creativity is a self-evaluative, iterative process as discussed by Csíkszentmihályi [14] (e.g., see Morris et al. [35]). Related is the model of a dynamic knowledge base [49] in which novel artefacts that have been evaluated as belonging to the domain are added to a system's set of exemplars, possibly altering the definition of the domain itself (e.g., as discussed by Boden [3]). Generate-and-check is another model that has been suggested as being representative of the creative process [47].

In considering the modeling of theoretical aspects of creativity, one particularly intriguing aspect that is often discussed is the tenuous balance that a creative system must maintain between novelty and typicality—the adherence to structural domain-defining rules

combined with an exploratory discovery of new, valuable artefacts. These two characteristics can sometimes seem at odds with one another; a creative system must both obey norms at some level and break them entirely at other levels. It is the juxtaposition of these qualities that evokes the perception of creativity: the observer recognizes and appreciates an artefact relative to its contextual domain while at the same time being challenged and surprised as a result of the artefact's unique traits and value. Csíkszentmihályi [14] emphasizes that creativity stems from a person learning the rules of and basic procedures of a domain and then channeling thinking based on those rules in new directions. Saunders and Gero [58] puts novelty and typicality on a spectrum called the Wundt curve or "hedonic function" and frames successful creativity in terms of finding the correct balance of typicality and novelty (see Figure 3.1). Margaret Boden [3], in her seminal work *The Creative Mind: Myths and Mechanisms*, compares (exploratory) creativity to navigating a "structured conceptual space" to find "things you'd never noticed before." Wiggins [66] elaborates a formal mechanism of Boden's concept of creativity by defining two rule sets, $\mathscr{R}$ and $\mathscr{T}$. Of these two sets $\mathscr{R}$ is a set of rules which "constrain the space" to a representation of "the agreed nature of what the artefact is, in the abstract"; $\mathscr{T}$, by contrast, is a set of traversal rules which, when constructed effectively, is designed to find concepts that have not been previously discovered. Ritchie [53], in defining empirical criteria for attributing creativity to a computer program, defines three essential properties, two of which are novelty and typicality (the third is quality, which Boden also emphasizes and which we will discuss below).

Many existing abstract frameworks for building creative systems have been described, several of which explicitly model the components of novelty and typicality (e.g., [65]). Our purpose is not to present a new framework or pattern for creative systems; rather our purpose is to discuss from an *implementation* standpoint how typicality and novelty can be modeled so as to explicitly leverage their unique contributions and simultaneously ensure that both are effectively achieved. In what follows we examine the suitability of a previously unexplored model in CC—a *constrained probabilistic model*—for this purpose. We describe how the

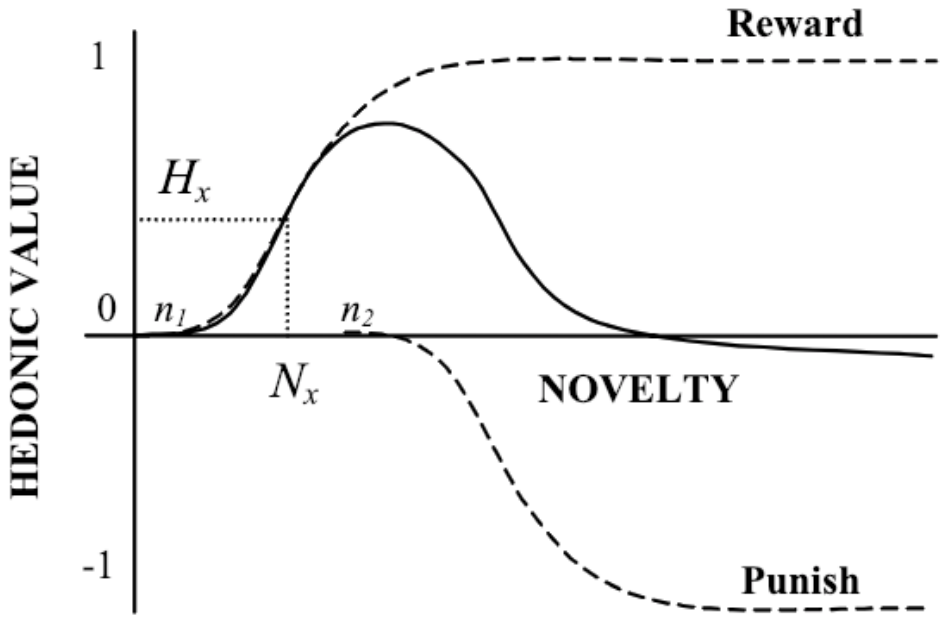


Figure 3.1: The Wundt curve models value as the sum of two nonlinear functions: H_x which rewards novelty, and N_x which punishes novelty beyond some threshold of typicality, from Saunders and Gero [58].

dual nature of this model mirrors the dual properties of typicality and novelty and how the model strikes an appropriate balance between them. As a concrete example of the effective application of these models to generate novelty and typicality, we describe an implementation of a constrained Markov model, NhMMonic, for generating mnemonic devices. We show using evaluative surveys that the system generates mnemonics that demonstrate typicality, novelty, and value (as measured by how well the mnemonic facilitates memorization and learning).

3.3 Parallels Between Computational Creativity and Constrained Probabilistic Modeling

Computational creativity can be thought of as a generative act in which, for some particular domain, the set of possible artefacts $\mathcal{D} = \{x_1, \ldots, x_n\}$ is represented as a random variable X that with probability $P(x_i)$ takes on the value x_i. The primary strength of probabilistic models is that they generalize well from a set of training examples to be able to generate

novel artefacts. Inasmuch as this generalization is accomplished independent of the biases of the system designer, it lends strength to the argument that probabilistic systems possess some degree of autonomy beyond manually-crafted rule-based systems. In practice, implementing a creative system in this manner presents two challenges.

One challenge is determining the probability distribution $P(X)$: with what probability should the model generate a particular x_i? This challenge can be solved explicitly—as in the case of systems that manually encode a generative process—or implicitly—as in the case of systems that attempt to learn abstract statistical properties from a set of training examples.

Prior to or in the course of resolving the first challenge, we face a second, more formidable challenge: defining the domain $\mathcal{D}$ itself. Decisions about whether a particular artefact x_j belongs or does not belong to $\mathcal{D}$ can vary from one individual to the next [31]. For now let us assume that D exists as a "fuzzy" subset of some larger domain, which we shall call $U_{\mathcal{D}}$ and which represents the universal set of all artefacts that can be represented using the same language with which artefacts in $\mathcal{D}$ are represented. For example, the domain of haiku exists as a subdomain of natural language generally. The domain of musical chorales exists as a subdomain of musical compositions generally. The fuzziness of the set $\mathcal{D}$ can derive from a variety of issues such as the difficulty in precisely defining $\mathcal{D}$ or the willingness of domain experts to accept artefacts that (to varying extents) break the rules typical of an artefact in $\mathcal{D}$.

Any particular creative system defines a set that more or less approximates $\mathcal{D}$ and possibly includes some artefacts that are less commonly agreed upon as belonging to $\mathcal{D}$ (see Figure 3.2). How this set is implemented is important in designing creative systems that efficiently generate artefacts in $\mathcal{D}$. For rule-based systems, the rules by which an artefact belongs within the set are hard-coded; logic is designed to prevent consideration of artefacts that break rules of the domain beyond some threshold. For evolutionary models, this set can be defined by designing a fitness function that penalizes artefacts outside of this domain.

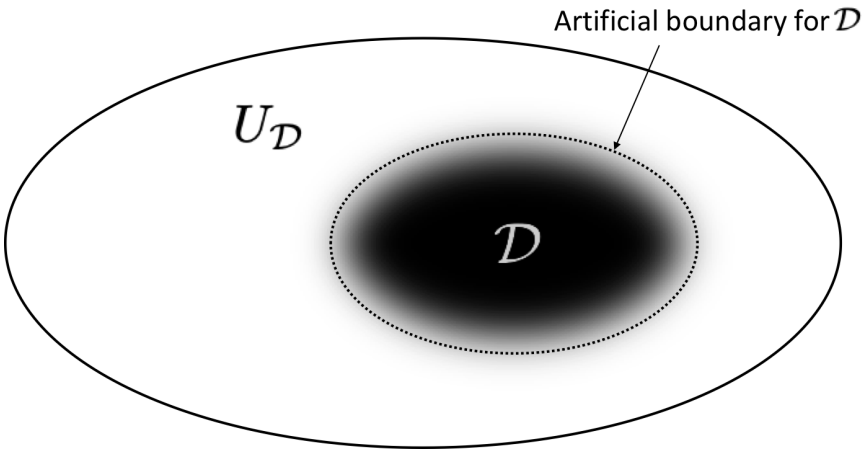


Figure 3.2: In many forms of creativity, the set of domain artefacts $\mathcal{D}$ exists as a structured subset of a larger domain $U_{\mathcal{D}}$ of all artefacts that can be represented using the same language as is used to describe artefacts in $\mathcal{D}$. Due to the inherent difficulty of defining belonging to a particular domain for a general audience, the set of artefacts included in $\mathcal{D}$ is in reality somewhat vague. In practice creative systems define a set that approximates $\mathcal{D}$ which defines the expressive range of the model. The extent to which this set includes or excludes artefacts that are commonly accepted as belonging to $\mathcal{D}$ controls how conservative or liberal the model will be in judging whether or not an artefact is representative of the domain.

The set can also be defined as a set of constraints given as input to a constraint satisfaction solver, but with limited sense of how good one solution is with respect to another [40].

In the process of generalization, probabilistic models trained with artefacts from $\mathcal{D}$ are typically capable of generating artefacts that do not belong in $\mathcal{D}$. Increased expressive power in these models (i.e., the ability to generalize novel solutions) derives from maximizing independence relationships between elements of an artefact (e.g., being able to model rhythm and pitch separately in a music composition). This process can, however, lead to the generation of artefacts whose combined elements produce artefacts that most would agree do not belong in $\mathcal{D}$.

Suboptimal solutions exist to ensure that a probabilistic model generates artefacts within the domain D of interest. Probabilistic models *could* ensure their output by minimizing independence assumptions (i.e., forcing the model to generate solutions more similar to the training data). This solution significantly decreases the model's ability to discover novelty

from the training data. This solution also requires training on data that is more precisely representative of $\mathcal{D}$. A second suboptimal solution is the generate-and-check or rejection sampling model: probabilistically generate artefacts using the over-generalized model and then filter results to those within the D [47]. This solution not only creates inefficiencies, but often assigns low probability to artefacts belonging to $\mathcal{D}$ [65]. In such cases it becomes improbable that the system generates valid artefacts in reasonable time [42].

A better solution to the problem of enforcing the model's domain of artefacts is the incorporation of constraints into a model that maintains probabilistic reasoning. The "fundamental entwinement of constraints and creativity" has been noted as an area of recent interest for creativity research, "with skillful and innovative handling of constraints seen as a prerequisite for apt creative performance" [40].

A constrained probabilistic model defines a set of rules for belonging in $\mathcal{D}$ as a set of constraints $\mathcal{C}$. Given $\mathcal{C}$ and a probability distribution $P_{U_\mathcal{D}}(x_j)$ for all artefacts in $x_j \in U_\mathcal{D}$, a constrained probabilistic model defines the probability of generating an artefact x_i as

$$P(x_i) \propto \begin{cases} P_{U_\mathcal{D}}(x_i) & \text{if } x_i \text{ satisfies } \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

By defining constraints explicitly, the model can be trained on artefacts from $U_\mathcal{D}$ generally, maintain independence assumptions that maximize expressivity, and ensure probability within the generative model is only assigned to artefacts which belong to $\mathcal{D}$.

There are several types of constrained probabilistic models including multi-valued decision diagrams (MDDs) for sequential domains [48]; MDDs that enforce constraints on non-discretized temporal sequences [56]; factor graphs for imposing constraints represented as regular languages Papadopoulos et al. [45]; and non-homogeneous Markov models [42]. Each model incorporates a probabilistic element designed to imitate statistical properties of a corpus—with model parameters (e.g., Markov order or context length) that control the degree of similarity to the corpus—and constraints to guarantee specifiable characteristics of

the application domain. Previous work has also shown how constraints can be used avoid plagiarism (i.e., limit the model's output domain to $\mathcal{D}$ less the artefacts used for training) [43]. It is of interest to note that much of the language used to describe the implementation of these models mirrors closely the language used to by creative theorists to describe the relationship between novelty and typicality. For example, Perez, Guillaume and Régin [48] describe the process by which the model generates new phrases as a "sampling of the solution set while respecting probabilities," specifying that the solution set "incorporate[s] some side constraints defining the type of phrases we would like to obtain."

3.3.1 Quality Assurance

We have discussed how constrained probabilistic models are well-suited for explicitly modeling typicality and novelty, but what about quality? As Boden [3] puts it, "a computer could merrily produce novel combinations till kingdom come. But would they be of any interest?" How well are constrained probabilistic models able to produce or evaluate *quality*?

To the extent that quality can be represented in either the system's probabilistic model *and/or* the system's constraint set, a constrained probabilistic model is naturally endowed with a function for evaluating the quality of the artefacts. By structuring the system's probabilistic model such that high quality artefacts (by some definition of quality) are assigned higher probability, the system will naturally gravitate towards stochastically generating artefacts of value (as will be shown in our demonstrative example). In cases where quality is a function of the presence or absence of certain characteristics (consider, for example, assessing quality based on the presence of satisfactory rhymes), the system's constraints can ensure that only artefacts of some minimum quality threshold are generated.

A constrained probabilistic model thus does not define its own function for evaluating quality, but *does* inherently encode one in the forms of probabilistic models and sets of constraints (both of which could be explicitly defined or themselves learned from some training data, as demonstrated in [4]).

3.4 Non-Homogeneous Markov Models

We describe a computational creative system for generating mnemonic devices using a non-homogeneous Markov model (NHMM), a constrained probabilistic model that is also called a constrained Markov model [42].

A Markov model $\mathcal{M}$ is a stochastic, probabilistic model defined over a finite state space that strictly adheres to the Markov property, meaning $\mathcal{M}$ is memory-less beyond a finite window. The set of all sequences $s = s_1, \ldots, s_n$ of length n generated by $\mathcal{M}$ is represented by S (in our current example this can be thought of as being equivalent to $U_{\mathcal{D}}$ from above). Every sequence $s \in S$ has a non-zero probability equivalent to

$$P_{\mathcal{M}}(s) = P_{\mathcal{M}}(s_1) \cdot P_{\mathcal{M}}(s_2|s_1) \cdots P_{\mathcal{M}}(s_l|s_{n-1})$$

$\mathcal{M}$ is constructed by computing the probability matrix $P_{\mathcal{M}}$ from training examples.

A non-homogeneous Markov model $\mathcal{N}$ is constructed from a Markov model $\mathcal{M}$, a sequence length l, and a finite sequence of unary constraints $\{C_1, \ldots, C_l\}$. The set of solutions for $\mathcal{N}$ is represented by S_C (equivalent to bounded $\mathcal{D}$ from above). With the constraints applied to $\mathcal{N}$, the probabilities of sequences generated by $\mathcal{N}$ must equal the probability of the same sequence generated by $\mathcal{M}$:

$$P_{\mathcal{N}}(s) = \begin{cases} P_{\mathcal{M}}(s) & \text{if } s \in S_C \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{N}$ initially constructs $l-1$ probability matrices identical to $P_{\mathcal{M}}$ in $\mathcal{M}$, one for each transition in the sequence to be generated. States or transitions that violate a constraint are removed. Arc consistency is then enforced on the probability matrices, meaning that states or transitions that do not lead to a solution $s \in S_C$ are removed (see Figure 3.3b). Because the probability matrices in the NHMM are arc consistent and therefore non-zero probabilities are guaranteed

to lead to a solution $s \in S_C$. This guarantee of solutions avoids the inefficiency generate-and-check where nearly all samples are rejected when the probability of a solution is small. Finally, the model is re-normalized such that probabilities $P_{\mathcal{N}}(s) = P_{\mathcal{M}}(s|s \in S_C)$ [42].

NHMMs have been applied to model music generation, generating melodies constrained to begin and end on the same note [42]. Barbieri et al. [2] apply NHMMs to generate lyrics matching rhyme, syllable stress, part-of-speech, and semantic constraints.

3.5 NhMMonic

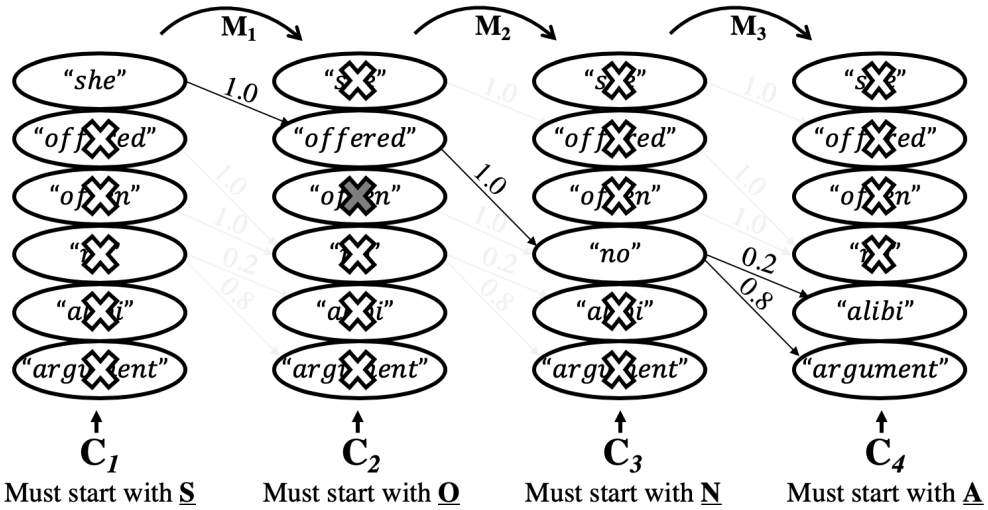
Here we demonstrate the application of constrained probabilistic modeling to computational creativity through *non-homogeneous Markov modeling of mnemonics* (abbreviated as NhMMonic). We define a *mnemonic task* as a sequence of words $s = s_1, \ldots, s_l$ to be memorized. A *mnemonic device* then is a sequence of words $m = m_1, \ldots, m_l$ of the same length generated such that for all $1 \leq i \leq l$ the first letters in the words s_i and m_i are constrained to be the same (see Figure 3.3). The primary purpose of a mnemonic device is to aid in memorization of the order and/or identity of a s by finding a more memorable sequence m that through its constrained similarity to s can serve as a reminder of s. The value of an artefact in this domain is heavily predicated on its effectiveness in facilitating memory.

To our knowledge no mnemonic device generation models have been formally presented. We find that most available Mnemonic generation tools online use what we will call a *template* method. The template method for mnemonic generation first determines a sequence of part of speech constraints as a function of the length l of the sequence to be generated. Words matching these constraints and the aforementioned first-letter constraints are randomly selected from a word bank to fit into the specific sentence structure. The shortcoming to most template-based methods is that they do not model transitions between words, resulting in phrases that exhibit grammatical cohesion, but not semantic cohesion.

Because NHMMs explicitly model transitions between words while allowing for constraints, we consider this model a good candidate for the mnemonic problem. Although

Stream-enterer, **O**nce-returner, **N**on-returner, **A**rahant

(a) A Mnemonic Task



(b) Constrained Probabilistic Model (NHMM)

"***S****he* ***o****ffered* ***n****o* ***a****rgument*"

(c) Mnemonic Device Generation

Figure 3.3: *The NhMMonic model.* (a) A mnemonic task (i.e., the four stages of enlightenment) to be memorized. (b) A non-homogeneous Markov model built to solve the mnemonic task. M_1, M_2, and M_3 represent Markov constraints; C_1, C_2, C_3, and C_4 denote unary constraints derived from the task. Nodes marked with white X's are removed due to violation of unary constraints while the node marked with a grey X is removed to keep the model arc consistent. Edge labels indicate transition probabilities. (c) A possible mnemonic generated by the model.

NHMMs can and have been used to impose part-of-speech constraints or templates, we chose not to include these constraints in our NHMM implementations preferring to demonstrate that even a relatively simple NHMM can provide good results. While we expect both models to be capable of generating novelty (or *uniqueness* as it is labeled in our survey), we expect NHMMs to outperform other mnemonic device models when it comes to the aspects of typicality relating to grammatical/semantic cohesion and ease of memorization.

3.6 Methods

In assessing the NhMMonic system we applied two variants of NHMMs. NHMM-1 has a Markov order of 1 and NHMM-2 has a Markov order of 2 (essentially treating each *pair* of words as a single state token). A higher Markov order allows the mnemonic output to more closely resemble the sample text, increasing the model's cohesion and typicality. A drawback of having a higher Markov order is that fewer solutions $s \in S_C$ are found and in some cases no solutions are found given finite training sentences. NHMM-1, with its lower Markov order, allows our system to find solutions when NHMM-2 does not.

For a mnemonic task $s = s_1, \ldots, s_l$, we derive a unary constraint oat position i to ensure that the first character of the sequence variable m_i matches the first given letter of s_i. For the purposes of improved readability of generated mnemonics we impose a few additional constraints. For NHMM-1, we constrain each sequence variables m_i to be at least 4 letters long and the last variable m_l to have ended a sentence in the training set. For NHMM-2 the only added constraint is to ensure that the last variable m_l is not a pronoun, preposition, conjunction, or determiner.

The code for the NHMMs used by the NhMMonic system are available in both a C++ implementation[2] (used for NHMM-1) and a Java implementation[3] (used for NHMM-2) online

3.7 Results

To evaluate the use of constrained Markov models for generating mnemonic devices, we devised an online survey to compare four different mnemonic device generation models:

- **Template**—a third-party model[4] that selects a part-of-speech template to match the desired sequence length and then randomly selects words matching part of speech and initial letter constraints from a hand-crafted word bank.

[2] `https://github.com/po-gl/ConstrainedMarkovModel`
[3] `https://github.com/norkish/downbythebay/tree/master/DownByTheBay/src/dbtb/markov`
[4] Available via `https://spacefem.com/mnemonics`

- **NHMM-0**—a model which randomly selects words matching initial letter constraints with probability derived from word frequencies in the training corpus.
- **NHMM-1**—a first-order NHMM as described above.
- **NHMM-2**—a second-order NHMM as described above.

The latter three models were trained on the COCA dataset [16]. NHMM-0 and NHMM-1 were trained on 6.8 million sentences from fictional works written between the years 1995 and 2015 while NHMM-2 trained on 3 million sentences from the same works.

Each model was used to generate 4 mnemonic devices for each of 19 different memorization tasks[5] (Figure 3.6 shows some examples of tasks included in the experiment). NHMM-2 was able to find satisfying solutions to 12 of the tasks.

To evaluate the generated mnemonics, we designed a survey in which each evaluation consisted of four parts:

1. The respondent was shown one of the 19 memorization tasks for 10 seconds.
2. The respondent was then shown a mnemonic device for the memorization task for 10 seconds (selected randomly from those generated by the four models).
3. The respondent was then given the (unordered) words from the original memorization task and asked to put them in the correct order based on his/her memory of the task and the mnemonic.
4. Lastly the respondent was asked to evaluate the mnemonic device (using Likert scales from 1 to 5) for
   (a) *memory*—ease of memorization
   (b) *flow*—grammatical/semantic coherence
   (c) *creativity*—overall creative value
   (d) *uniqueness*—degree of novelty

[5]Mnemonics for all models can be seen at `https://tinyurl.com/yxczxjh7`

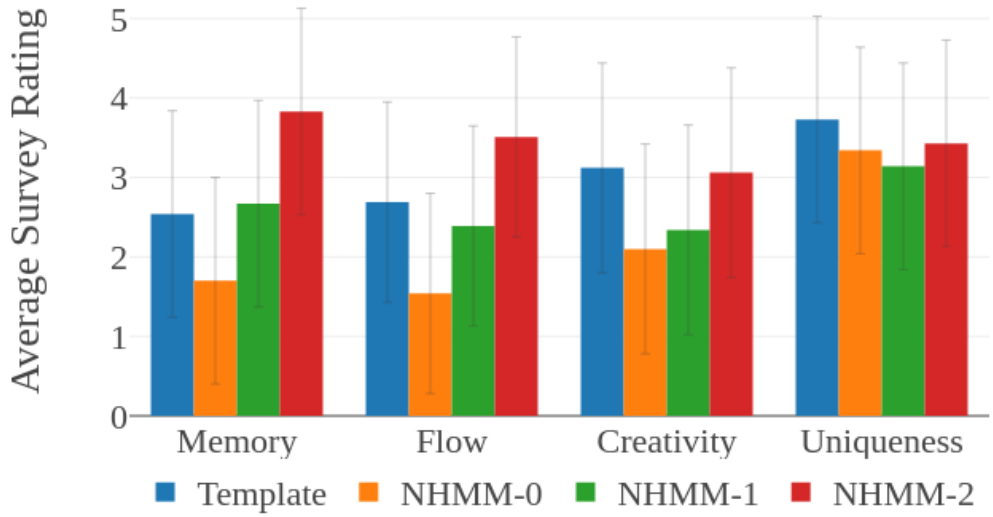


Figure 3.4: *Survey Results*. Average ratings from 320 evaluations across four metrics for four different mnemonic device generation algorithms. Error bars are standard deviation. The ease of memorization of mnemonics from the NHMM-2 model appears to be associated with improved flow with respect to other models.

Each respondent completed four evaluations in this manner.

A total of 80 individuals completed the survey for a total of 320 mnemonic device evaluations. The survey was distributed to different social media websites, such as Reddit, Facebook, and Twitter. No personal information was gathered before or after the survey was taken. Figure 3.4 shows average scores for the four evaluated characteristics by model. The NHMM-2 model made notable improvements over other models in the categories of ease of memorization (*memory*) and grammatical/semantic cohesion (*flow*). Although the NHMM-0 model performed relatively poorly on *memory*, *flow*, and *creativity*, this model was considered equally capable of generating novelty (i.e., *uniqueness*).

Figure 3.5 shows the impact of task length on ease of memorization, showing generally that the longer a mnemonic task is, the more difficult mnemonics generated for the task are to remember. The graph also shows, however, that the NHMMs and NHMM-2 in particular, is able to generate mnemonics that maintain ease of memorization even for longer tasks.

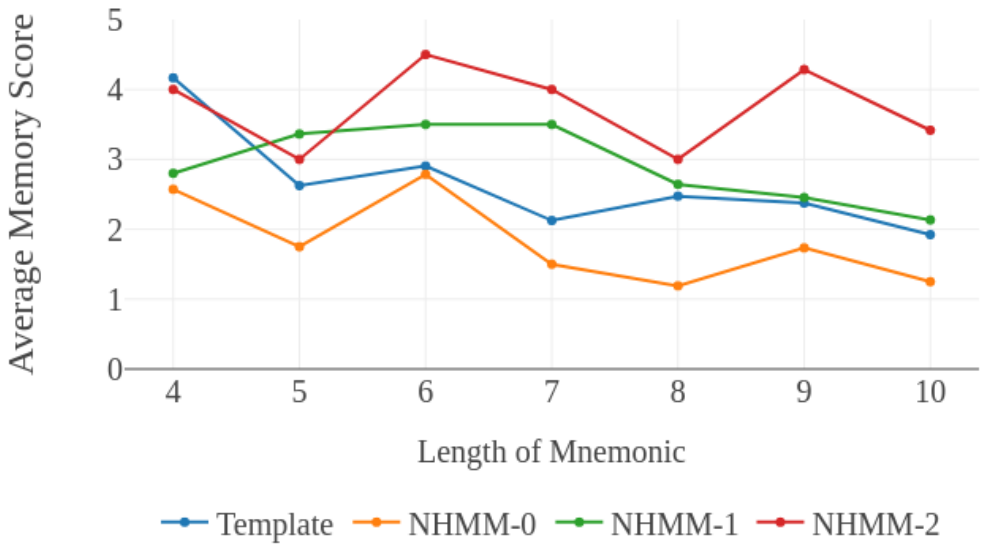


Figure 3.5: *Impact of Task Length.* As the length of the memorization task increases, the effectiveness of mnemonic devices decreases across all models, but at a much lesser rate for the NHMM-1 and NHMM-2 models. We hypothesize that this is owing to the sustained grammatical and semantic flow that these models achieve from the constrained Markov model.

Figure 3.6 shows seven mnemonic device tasks together with the highest-rated mnemonic devices (as per average memory score) generated by NhMMonic for the task.

3.8 Discussion

Survey results demonstrate that increased grammatical/semantic cohesion afforded by probabilistic Markov models are associated with gains in ease of memorization. The fact that increasing the Markov order leads to further gains in both *flow* and *memory* is further evidence of this correlation. These gains from increasing the Markov order were also mirrored in increased creativity scores, suggesting that in the domain of mnemonic device generation, there is an association between the creative success of a mnemonic device and how easily it can be remembered.

This association between the success or popularity of an artefact and the ease with which the brain is able to process and remember it has been observed in creative domains that do *not* deal directly with memorization tasks. A notable example is the study by Nunes

Four Stages of Enlightenment: Stream-enterer, Once-returner, Non-returner, Arahant
"She offered no argument" (NHMM-2, 5.0)

Dante's 9 Circles of Hell: Limbo, Lust, Gluttony, Greed, Anger, Heresy, Violence, Fraud, Treachery
"Lovely little girl giggles as his voice for them" (NHMM-1, 5.0)

Last 10 Winners of the FIFA World Cup: France, Germany, Spain, Italy, Brazil, France, Brazil, West Germany, Argentina, Italy
"Four-year-old grandson she is bumped from behind with an inflection" (NHMM-2, 4.0)

First 9 ICCC Locations: Lisbon, Mexico City, Dublin, Sydney, Ljubljana, Park City, Paris, Atlanta, Salamanca
"Like most days she looked pretty puny and sickly" (NHMM-2, 4.5)

Stages of Grief: Denial, Anger, Bargaining, Depression, Acceptance
"Dreams about being dragged against" (NHMM-1, 5.0)

Levels of Biological Organization: Biosphere, Ecosystem, Community, Population, Organism, Organ System, Organ, Tissue, Cell, Molecule
"Blue eyes could pick out one of those clownish men" (NHMM-2, 4.5)

Cell Mitosis Cycle: Interphase, Prophase, Prometaphase, Metaphase, Anaphase, Telophase, Cytokinesis
"I pushed past me and the career" (NHMM-2, 5.0)

Figure 3.6: *Top-rated mnemonics generated by NhMMonic.* Seven mnemonic device tasks are shown. Each task consists of a description (bold and underlined) followed by a list of words requiring a mnemonic device. Below each task is the NhMMonic-generated mnemonic device that received the highest memorization score (with the exact model and score given in parentheses).

et al. [38] that demonstrates an association between the popularity of music and the degree of repetition in the song. Researchers observed that increased repetitiveness contributed to higher "processing fluency", meaning the ease with which the brain is able to grasp a new concept or artefact. A constrained Markov model, through its probabilistic transition model, naturally assigns higher probability to frequent word transitions (which we might assume have higher processing fluency) while using constraints to ensure that generated mnemonics also satisfy the basic requirements of a mnemonic device.

As is typical of Markov-based models, increasing the Markov order can also have negative consequences. The higher the order the more similarity exists between generated artefacts and the training data. Increasing the order also increases the likelihood of the model not being able to find a solution that satisfies both the (now more stringent) Markov constraints and non-Markovian constraints. Both of these problems can be overcome by

training on more training data, but the amount of training data needed to sufficiently eradicate the problem increases exponentially with the Markov order.

Independent of the model training, some mnemonic tasks are inherently more difficult owing to the low frequency of words and word beginning with certain letters (this is, of course, language-specific). Consider for example trying to devise a mnemonic device in the English language for the first five dynasties of China, "Neolithic, Xia, Shang, Zhou, Qin". Solutions certainly exist, but unless the model sees examples in training of word pairs that would be suitable for each word pair in the task (less likely for infrequent collocates), the model will not be able to find them. On these types of tasks we might expect the non-Markovian models to perform better.

We considered other variations of constraints that might have further improved the results of our model. One improvement considered was to constrain more than just the first letter of each word in the mnemonic to match the task. We thought this might further increase the ease of memorization. However, it is generally the case that as constraints become more strict, the model is able to find fewer solutions, often leading to the model being unable to find satisfying solutions. Another improvement we considered was combining the Template and NHMM approaches through part of speech constraints in the NHMM model. We also considered ways to impose semantic themes within mnemonic devices either through unary semantic constraints or through varying the training data. We leave these as exploratory ideas for future work.

Many forms of creativity have relational structure (e.g., rhyme schemes, repeated motifs, etc.). Unlike the example we have shown here which uses solely unary constraints, relational structure is most effectively realized using binary constraints. Sampling from constrained Markov models with binary constraints is known to be a much harder problem (see [54]), however recent work has been done towards providing reasonable solutions [45, 56]. This has relevance for imposing semantic constraints in models of mnemonic device generation because binary constraints can effectively be used to impose *floating constraints* (i.e., constraints that

can be satisfied at variable positions) rather than specifying a specific word position where semantic constraints must be satisfied.

NHMM doesn't directly model all aspects of creativity. For example intention, explicit self-evaluation, others? Constraints themselves can be learned or imitated. One ramification of learned constraints is that in addition to whatever constraints are required to define typicality, additional constraints could themselves be probabilistically applied in generating artefacts. This would allow constraints to be "broken" (or rather never applied) with some degree of probability, demonstrating a method by which rules can be "intelligently" broken.

In this work we have discussed aspects of constrained probabilistic modeling that are well-suited for consistently generating novelty and typicality in computational creative artefacts. As an example, we have demonstrated the application of non-homogeneous Markov models to the problem of mnemonic device generation. Our results suggest that the constrained Markov model approach is able to effectively generate mnemonic devices that satisfy basic requirements of mnemonic devices while exhibiting elevated levels of grammatical/semantic flow, ease of memorization, and creative value.

3.9 Acknowledgements

Many thanks to the team at spacefem.com for their assistance using the spacefem mnemonic device generator.

Chapter 4

A Leap of Creativity: From Systems that Generalize to Systems that Filter

This paper was presented at ICCC 2020 and published in the conference proceedings pp. 297–302

Abstract

In his work "Mere Generation: Essential Barometer or Dated Concept?", Ventura [64] categorizes creative processes along a spectrum of increasing creativity. While the spectrum provides insight into the dimensions through which creativity can be augmented, it does not of itself provide insights into how to advance a system through these dimensions. In this paper, we present some theoretical and practical insights on advancing along one commonly problematic rung of this ladder, namely from a system that exhibits *generalization* (i.e., the ability to generalize beyond an inspiring set) to a system that exhibits *filtration* (i.e., the ability to self-evaluate and filter results). One potential challenge in this transition is that filtration requires having a sufficiently large number of solutions to filter from the generalizing model. We propose that one solution to this problem is achieved not through increasing the size of the inspiring set (an obvious solution that brings additional problems), but rather through amplifying the generalization of the system to produce a greater set of novel artefacts to filter. We compare a new version of a system, NhMMonic, for generating creative mnemonic devices with a new conceptualization model that allows greater generalization. We demonstrate how filtration, which was not possible in the early version of NhMMonic, only becomes feasible with the more generalizable model.

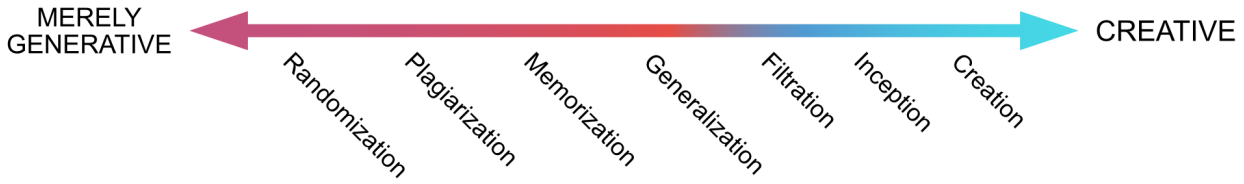


Figure 4.1: Ventura's spectrum of creative systems provides a means by which to measure the progress of a system towards becoming creative [64]. Characterizing challenges and solutions that are specific to each level in the spectrum helps to actualize the spectrum into becoming a guide for building more creative systems.

4.1 Introduction

The field of Computational Creativity (CC) has been supported in its quest by several significant contributions in the domain of CC theory. One such contribution exists in Ventura's spectrum of creative systems [64]. This spectrum suggests that there exist at least seven different levels along the path towards computational creativity including levels such as randomness, memorization, generalization, and filtration (see Figure 4.1). Ventura asserts that along this spectrum, real computational creativity starts at least as early as generalization with filtration representing perhaps a conservative threshold.

While this spectrum is useful for measuring the progress of applied CC systems, it leaves two important questions unanswered:

1. *For each level of the spectrum, what challenges are CC systems likely to encounter?*
2. *What suggestions can be made to overcome those challenges?*

Answers to these questions would provide a way to actualize the spectrum into a guide for augmenting the creativity of computational systems.

Our motivation in considering these issues came about in the context of our previous work using constrained Markov models to generate mnemonic devices [5]. Markov models are an example of a generalizing model. The application of constraints to Markov models

represents the act of filtration. In applying constraints to generate mnemonic devices, it frequently occurred that no satisfying solutions could be found.

The purpose of this paper is to provide answers to two questions stated above with specific regard to systems that have achieved the level of *generalization* and are attempting to make the "leap" to the level of *filtration*. This step is of interest as it marks the transition from a budding creative system to an intentionally creative system. This leap is significant in light of the fact that of the last four levels of the spectrum—where true creativity is said to emerge—this is the first step.

Generalization systems produce artefacts using an internal *conceptualization*—a model which embodies an understanding of a domain and allows for the creation of artefacts that belong to the domain [65]. Examples of conceptualizations include using long short-term memory models for music generation [36], neural networks for visual art [37], and Markov processes for music and text generation [2, 42].

One particular challenge we have repeatedly observed in the development of CC systems at this level is the challenge of dealing with diminishing solution spaces. This problem arises commonly when attempts are first made to add filtration to a generalization system because filtration by definition implies the reduction of a system's solution space. The purpose of the filtration step is to equip the system with self-evaluative capabilities for restricting the artefacts it generates based on measurements of fitness. However, a well-known trade-off arises: stricter filtering leads to better, but fewer results. In some cases the results are so few that it becomes difficult to justify that the system is capable of generating anything, let alone artefacts that are novel. How can systems overcome this challenge?

A simple solution for increasing the solution space is to simply increase the size of the inspiring set. For many conceptualizations of CC systems this alone will increase the overall throughput of the system, and often increases the generalizability of the system as well. However, for most domains, finding a larger inspiring set ranges from being impractical to an impossibility. What more practical solutions exist?

We propose and illustrate through example how increasing the generalizability of a generalization system through abstraction and regularization can increase the solution space without requiring a larger inspiring set. Well-known methods exist for generalization of most conceptualization models used for CC systems, including L1 and L2 regularization for neural networks, shortening the Markov window length in Markov processes, generalizing the fitness function for genetic algorithms, and abstracting rules in rule-based systems. Through regularization and abstraction, a system is able to better leverage the knowledge in an inspiring set in order to increase the solution space.

In demonstrating the impacts of abstraction and generalization, we comparatively consider the performance of two models: a less abstract model (CoMP) and a more abstract model (CHiMP). We assess the ability of each model to intentionally produce *novel* artefacts. We choose to focus explicitly on the creative attribute of *novelty*—setting aside the attributes of value and intentionality—inasmuch as it is the attribute of creativity most directly relevant to our discussion [53, 64]. We discuss the impacts of generalization on value in the discussion section below.

4.2 Methods

NhMMonic [5], is a CC system designed to generate mnemonic devices. At its heart, NhMMonic uses a constrained Markov process (CoMP) for its conceptualization model. This constrained Markov process allows for the combination of a (non-hidden) Markov process (e.g., trained on words) and a set of unary constraints (e.g., word-starts-with constraints) such that the model is able to generate constraint-satisfying sequences according to Markovian probabilities [42]. In previous work we demonstrated through qualitative surveys the strength of this model (particularly at higher Markov orders) for generating effective mnemonic devices. A byproduct of our analysis revealed that for many mnemonic device problems, the addition

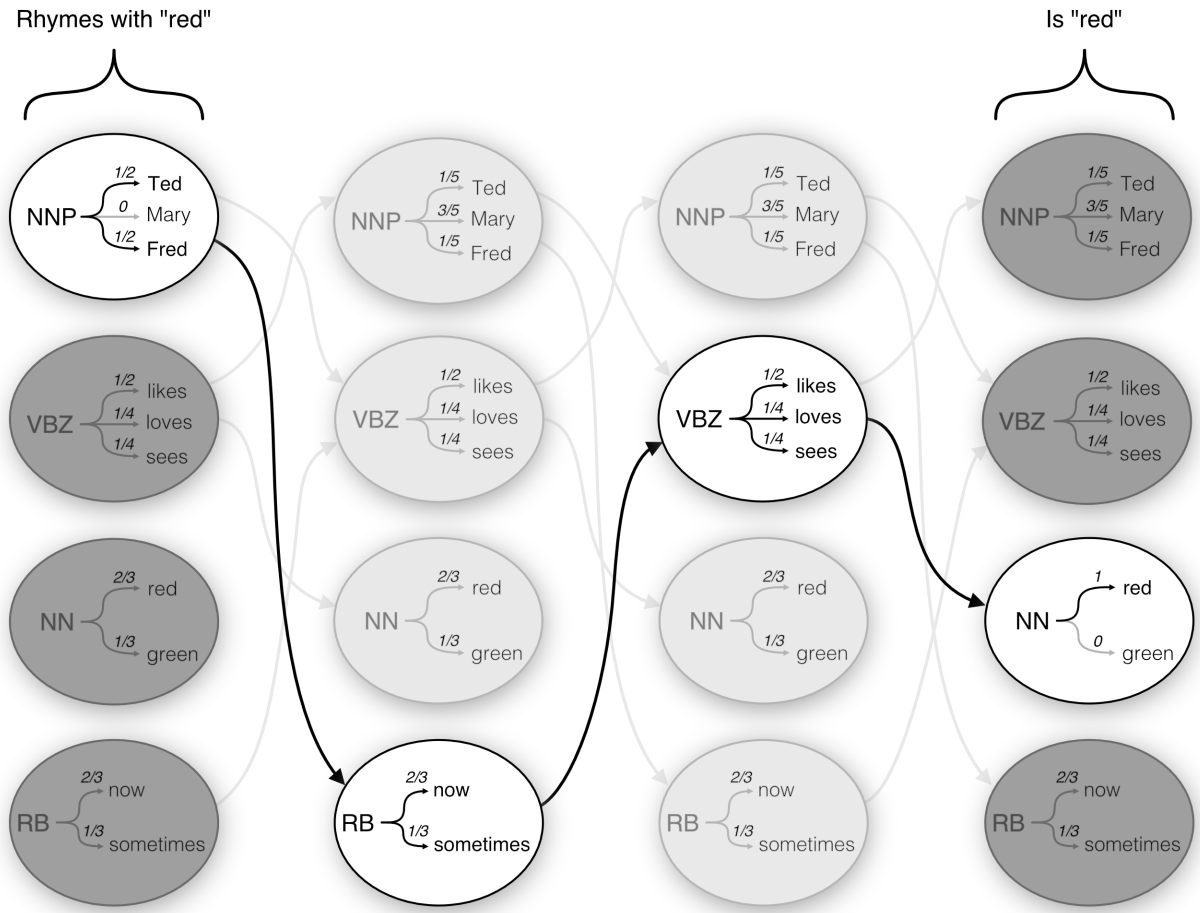


Figure 4.2: A high-level schematic of a constrained hidden Markov process (CHiMP) of length 4 constrained so that the last word is "red" and the first word rhymes with "red". Each column represents a position in the sequence to be generated. Each node represents a hidden state (i.e., part-of-speech) and a probability distribution for the observed states (i.e., words) that can be generated from that hidden state. By pruning observed states that are disallowed by constraints and then adjusting probabilities to maintain arc-consistency, the resulting model generates constraint-satisfying solutions with probability relative to the original probability distribution [22]. Hidden states pruned directly from applying constraints are indicated by dark grey nodes and states pruned during arc-consistency are indicated by light grey nodes.

of constraints (i.e., filtering) resulted in NhMMonic being incapable of finding satisfying solutions despite being trained from relatively large inspiring sets.

A known method for increasing the generalization of Markov models is through the introducing of an abstract hidden layer resulting in a model known as a *hidden* Markov process. Direct dependencies between adjacent observed sequence elements are dissolved in the hidden Markov process, allowing for greater decoupling between sequence elements. This generally results in hidden Markov processes having significantly higher expressivity with respect to their *non-hidden* counterparts.

To combat the challenges facing NhMMonic with respect to a diminishing solution space, we designed a new conceptualization model for the system that combines hidden Markov processes with constraints in much the same way that constrained Markov processes combined *non-hidden* Markov processes with constraints [22]. The resulting model is called a constrained Hidden Markov process (CHiMP) which is visualized in Figure 4.2. The CHiMP model was chosen under the hypothesis that increased abstraction, resulting in increased generalization, would lead to a significantly larger solution space.

In implementing a filtration system, it is apparent that a large solution space is needed. Using two hypothetical models A and B (seen in Figure 4.3) we illustrate the restriction that solution space imposes on a system's ability to step from a generalization system to a filtration system. Model A fails to have a solution space after filtering and thus remains a conceptualization for a generalization system. Model B, however, has a larger beginning solution space β due to an increase in the model's ability to generalize the inspiring set. Thus model B has a usable solution space β' after filtering and can be categorized as a filtration system.

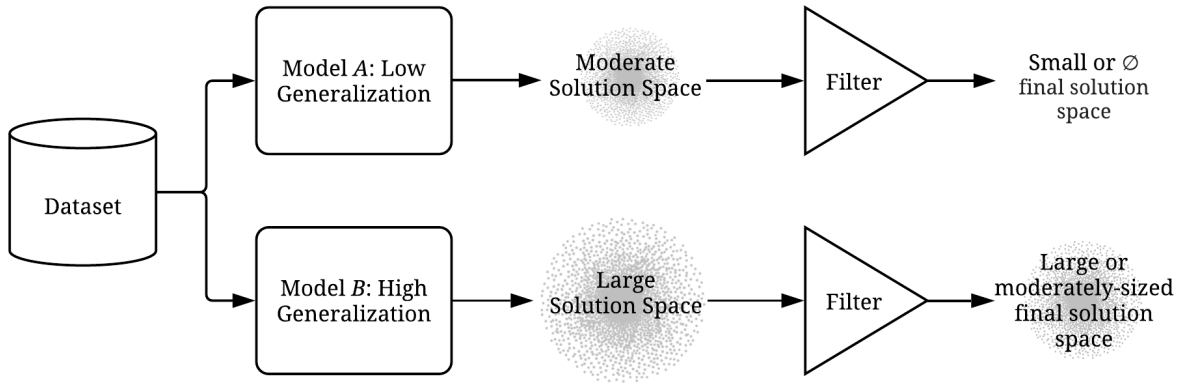


Figure 4.3: The application of filters on two hypothetical models (A and B) demonstrates the requirement for larger solution spaces (increased generalization) in order to endure filtering with a usable solution space. Model B has a usable solution space after filtering; thus the model has moved further along in the spectrum from generalization to filtration.

4.3 Results

In demonstrating the increased generalization (and hence increased solution space) of CHiMP over CoMP, we compared the results of each model trained on the Corpus of Contemporary American English (COCA) [16] and provided the same set of constraints. In particular, we selected training sets from the 2012 fiction portion of COCA and constrained each model to only output sequences in which the first letter of each word began with the same letter (e.g., a tongue-twister). We chose this problem because it represents a fairly general example of constrained sequence generation that is easily adapted to sequences of varying lengths. Results are averaged over 26 instances of the problem with each instance having constraints defined with a different letter of the English alphabet.

Some qualitative results are shown in Figure 4.4. It should be noted that within the subset of 40 sequences generated by CHiMP, no duplicate or similar solutions where present; whereas 6 sequences were duplicates (or very similar) in the subset generated by CoMP.

We examined the effect of changing the sentence/model length on the novelty of the system in terms of the total number of unique solutions capable of being generated by each model (see Figure 4.5). As the sentence length increases, so too do the number of constraints

on the sequence to be generated. In the abstracted CHiMP model, this is inconsequential; the model can afford to make restrictions at the observed node that do not affect transitions between sequence positions (which are isolated in the hidden layer). Only occasionally do a sufficient number of pruned states combine to require the pruning of a hidden state node, but such is a relatively rare occurrence.

By contrast, the effects of increased sentence length on the CoMP model are severely limiting. Each added position would typically add a number of novel unique solutions *if it did not come with the addition of a new constraint.* The newly constrained position has direct influence on previous observed sequence states and thus pruning values from the domain of these variables directly results in the removal of transitions between adjacent sequence positions. This results in a relatively slow growth in the solution space as sentence length grows.

The increase in the CHiMP model appears to be exponential owing to the multiplicative effect achieved by maintaining large domains for adjacent variables in the hidden layer.

Similar trends in the impact on novelty are manifest when we vary the training set size, keeping sentence length constant (see Figure 4.6). We see that the size of the solution space for the CHiMP model increases exponentially. The CoMP model also appears to have some slightly exponential growth, but at a significantly lower rate. This is again what we would expect to see. Increasing the training set size (when such is a possibility) still has a more significant impact on CHiMP than on CoMP model.

The results shown in Figures 4.5 and 4.6 suggest that CHiMP, with respect to CoMP, facilitates exponentially more novelty. The solution space of the CoMP model is by definition a subset of the solution space of the CHiMP model, and for most training and constraint sets will be a substantially smaller subset. It is expected that of the novel results produced by CHiMP, some will have higher value than the solutions shared by both models. Because the CHiMP model abstracts to a more significant degree from the training set than the CoMP model, we might expect a greater portion of the novel solutions to be of lower value. The

CoMP Tongue Twisters:

late last light levels like lady
Diaz did dinosaurs died dell drove
max mowed my mother made my
language lessons last look little lamb

CHiMP Tongue Twisters:

queen Quanhe quite quiet queasy qualified
flower facing forward for from forester
free feeling facing followed free fate
every educated Elizabeth expected Erika enchanting

Figure 4.4: Example results from generating 6-length tongue twisters (i.e., alliterative constraints) from both the CoMP and CHiMP models. Both models were trained on 10K sentences. Results are chosen from a randomly selected subset of 40 sequences from each model. The quality of tongue twisters is roughly equivalent between both models (both poor), but the CHiMP model is capable of generating exponentially more solutions. This suggests that increasing the Markov order in the CHiMP model (as an example of more stringent constraints) will have far less deleterious affects on the solution space as compared to a similar increase in the CoMP model.

suggestion from qualitative results shown in Figure 4.4 is that there is no obvious degradation of value. However, we do not currently have results to fully assess the extent to which value degrades (or doesn't). In any case the expressivity of the CHiMP model enables a simple solution: introduce new or stricter filtering by increasing the number and stringency of constraints.

4.4 Discussion and Conclusion

In progressing from a generalizing system to a filtration system, our results provide meaningful insight into two important questions relating to Ventura's spectrum of creative systems:

1. *For the filtration level of the spectrum, what challenges are CC systems likely to encounter?*

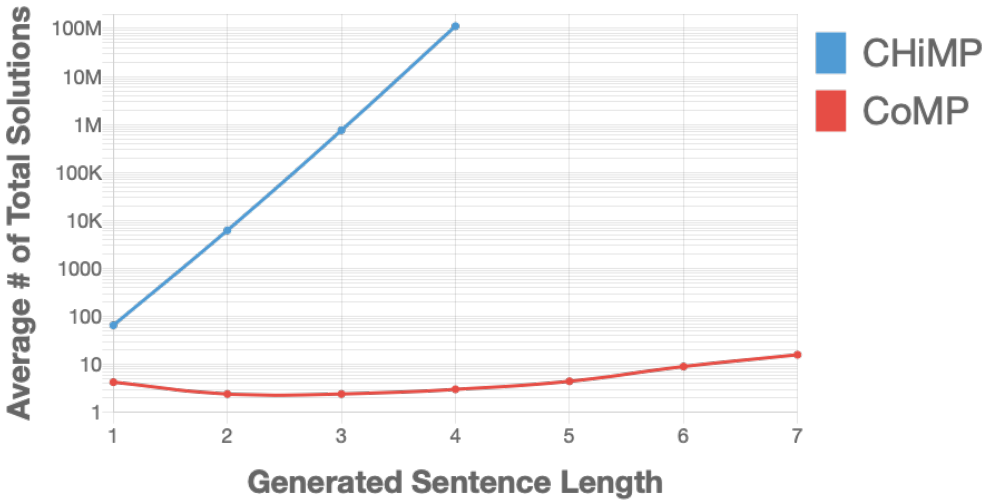


Figure 4.5: The effects of sequence length on the number of total solutions generated by each model with a fixed training set size of 300 sentences. Both models are constrained such that each word in a sequence starts with the same letter; counts of total solutions are averaged over 26 runs (each run using a different letter from the English alphabet). We see that as the sequence length increases, total solutions for the CHiMP model increases exponentially (given the logarithmic scale) whereas the CoMP model stagnates.

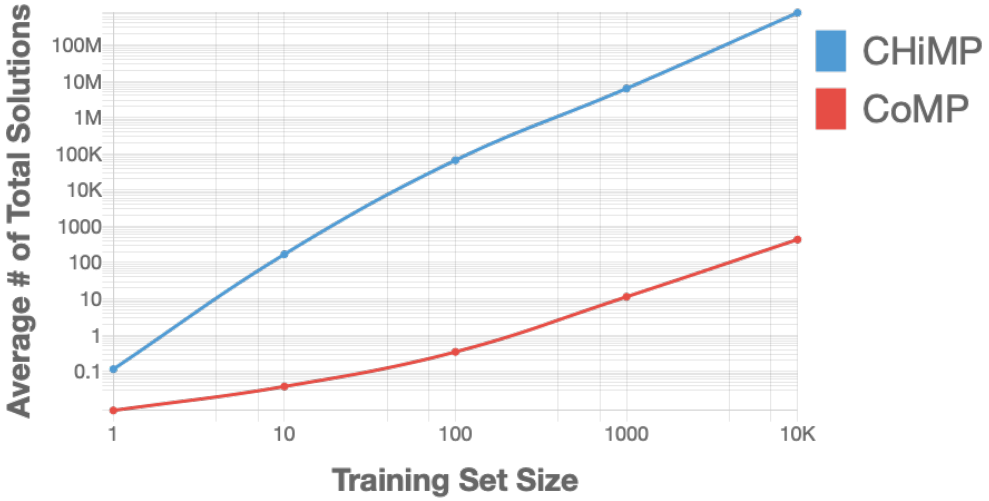


Figure 4.6: The effects of training corpus size (number of training sentences) on the number of total solutions generated by each model with a fixed sequence length of 3. Both models are constrained such that each word in a sequence starts with the same letter; counts of total solutions are averaged over 26 runs (each run using a different letter from the English alphabet). The total solutions of both models increase in an almost parallel way; however, at 10K training sentences, CHiMP well exceeds 100M total solutions which contrasts CoMP at 1000 total solutions.

2. *What suggestions can be made to overcome these challenges?*

A significant challenge for CC systems attempting to transition to a filtration system is as more constraints (or filters) are put on the system, the solution space diminishes to the point of being too small to filter. As demonstrated in the CoMP model (Figure 4.5), the insufficient solution space prevents being able to apply more constraints and filters to produce higher quality artefacts.

The problem is not specific to our results or to Markov models. Filtering, by nature, reduces the solution space. As shown in Figure 4.3, any CC system with low generalization may fail to have a usable solution space after filtering.

Greater generalization can address the aforementioned problem. We see from our results that our model with greater generalization, CHiMP, excels in solution space size even as constraints are added (see Figure 4.5). The primary difference between CoMP and CHiMP is an added layer of abstraction in CHiMP that affords greater generalization. The solution to a diminished solution space is to increase the level of abstraction in the model. This increases the generalization ability of the model and results in a solution space substantial enough to "survive" filtering.

Increased constraints allow for greater creativity and quality because the system can use constraints to explicitly articulate and enforce the system's goals and intentions. For example, in Markov models, increasing the Markov order (a form of adding more constraints) significantly improves the coherency of natural language, but the solution space is heavily diminished. With the CHiMP model, the solution space is sufficiently enlarged to avoid these devastating consequences to the solution space. Besides changes to the Markov order, other possibilities open up for using constraints to filter results to further improve quality, including semantic constraints, structural constraints, and even more complex *n-ary* constraints. It is also often the case that constraints can be easily described in human-interpretable language, enabling the system to provide framing for its creative behavior, contributing to an increased perception of creativity in CC systems [11].

It is important to acknowledge the negative consequences of increasing the generalization in a learning model. In particular, generalization decouples dependencies between variables which can result in a loss of information during variable assignment. For example, generalizing to a hidden Markov model takes a significant toll on language coherence. In short, the novelty achieved by generalization comes with a trade-off in value. We hypothesize that this deterioration can be offset in the application of filters to preserve the information lost. We plan to examine this issue in future work.

Through developing a system (CHiMP) that more effectively achieves filtration, we have discovered insights into the challenges present in the leap from *generalization* to *filtration* and how to overcome them. The challenge of diminishing solution spaces can be overcome by amplifying the generalizing ability of the system through abstraction. Having realized the leap from generalization to filtration, the community is now poised to address the challenge of making the subsequent leaps along Ventura's spectrum of creative systems, advancing past filtration into *inception* and ultimately *creation*.

Chapter 5

Probabilistic Generation of Sequences Under Constraints

This paper was presented at i-ETC 2020 and published in the conference proceedings pp. 135–140

Abstract

There is growing interest in the ability to generate natural and meaningful sequences (e.g., in domains such as language or music). Many existing sequence generation models, including Markov and neural algorithms, capture local coherence, but have no mechanism for applying the structural constraints that are so often essential for the development of meaning. We describe a novel solution to this problem which combines hidden Markov models with constraints, allowing sequences which obey user-defined constraints to be generated according to data-driven probability distributions. Compared to other constrained probabilistic solutions, our Constrained Hidden Markov Process (CHiMP) has significantly greater expressivity, allowing the user to generate constrained sequences that are longer and which have more numerous structural constraints.

Source code is available at *https://github.com/brandonbiggs/Chimp*

5.1 Introduction

Sequence generation is a common task in the field of artificial intelligence, particularly for domains such as music, procedural content generation, natural language generation, etc. Models used to generate sequences are trained on a corpus and are then iteratively sampled

using some probabilistic distribution to generate a sequence of words. Many generative models (such as n-gram, Markov, and recurrent neural models) use previously generated tokens in the sequence to influence which future tokens are generated; however, they make no guarantee as to what a particular token will be [23, 51]. The stochastic nature of these models is desirable for many types of problems where generalization beyond the training corpus is essential to the generation of novel and yet coherent sequences.

Stochasticity becomes problematic, however, when needing to generate sequences whose meaning relies on structure. Music, for example, progresses sequentially; notes are followed by more notes. However, to elevate a musical phrase to be something interesting, higher features such as motifs need to be present in the sequence [1, 38]. Purely stochastic models have no way of ensuring that a motif or repeated pattern is generated in a longer musical phrase. Likewise coherent English sentences require proper subject-verb and noun-pronoun agreements, possibly between distant sequence positions. Quality generation of English sentences are important to any natural language application, but are increasingly in demand for applications such as virtual assistants or procedural content generation [32].

A common solution to the lack of structure in sequential data models is to combine these models with constraints. Constrained Markov processes in particular have been very successful at imposing structure when combined with constraints [2]. These models are capable of imposing structure in the form of unary constraints. For example, rhymes can be created by constraining words at different positions to belong to the same rhyme group. They have also been used in music generation [42] and to constrain against plagiarism [43].

A significant and well-known drawback of constrained Markov models (and constrained models in general) is the inability to find satisfying solutions when constraints become numerous. This is due largely to the high coupling between states in a Markov process which causes changes in the state space at one position to directly and significantly affect the state space at neighboring positions. As more constraints are added, the diminishing state space problem is compounded to the point where the model is not able to find sequences that

satisfy both the Markov constraints (i.e., state sequences allowed by the Markov transitions) and the unary constraints.

To address these shortcomings, we propose a novel solution to the constrained sequence generation problem that uses a constrained hidden Markov process (CHiMP) model. We demonstrate how the increased expressivity that emerges from hidden Markov processes with respect to non-hidden Markov processes can be similarly applied to a constrained Markov model in order to vastly increase the solution space for any constrained sequence generation problem. Lastly we provide a comparative analysis of these two models on a real-world sequence generation problem as a demonstration of the improved expressivity of the CHiMP model over other constrained Markov processes.

5.2 Related Work

Markov models (which are also very similar to n-gram models) are ideal for sequence generation; however, other types of models are commonly used for sequence generation. Neural networks in the form of recurrent neural networks (RNNs) can be used to repeatedly predict (i.e., generate) words in a sequence. RNNs differ from Markov models in that they are "fuzzy", meaning they perform higher-dimensional interpolation between training instances for their predictions [23]. This allows RNNs to synthesize and use their training data in a more complex way compared to Markov models. However, the fuzzy aspect of RNNs means that the model makes no guarantees about a generated sequence. For a generated English sentence, an RNN may exhibit a comprehension of a complex interaction present in the training data, but the longer sequence generated fails to satisfy subject-verb agreement. Combining RNNs with constraints is thus an area of ongoing research [24].

Long Short-term Memory (LSTM) are a variant of RNNs designed specificially to maintain a longer term memory about tokens that the model has previously generated [23]. LSTMs use multiple neural networks and memory cells to achieve this increased ability to remember inputs over longer periods in a sequence. This allows LSTMs to display a better

understanding of lasting structure and patterns in a sequence over RNNs. An LSTM will have an easier time replicating a desired larger structure such as subject-verb agreement, however the model does not guarantee such structures will emerge.

Markov processes generate sequences by looking at the previous state (or previous n states for an n-order Markov model) and randomly sampling according to trained transition/emission probabilities to generate a new token. The process is iterated to generate a full sequence of tokens. The simplicity of these models lends to them being easy to implement and efficient to run [42]. Examples of Markov implementations include systems that react interactively to music input [41] and text-to-speech synthesis where speech waveform generation is generated via a hidden Markov model [62].

When generating sequences, Markov processes produce sequences that share a common style with the data the model was trained on. In the context of natural language synthesis, the style sharing properties of Markov processes can be exploited to change speaker identities, emotion of the speech, and the cadence of the speaker [62].

The defining feature of a Markov process is that it adheres to the Markov property which is that the next state in the sequence is determined only by the previous state to it, i.e.,

$$p(s_i|s_1, \ldots, s_{i-1}) = p(s_i|s_{i-1}).$$

In the context of sequence generation, the Markov property is well-suited to domains such as music where, aside from higher features, the next note relies on the previous note (i.e., music exhibits Markovian aspects) [7].

Previous efforts have been made to combine probabilistic models with constraint satisfaction. Pachet et al. introduce constrained Markov process (which we will refer to as CoMP) as a method for applying user-defined constraints to a non-hidden Markov model [42]. Likewise, factor graphs combine a non-hidden Markov process with an automaton to create a system functionally equivalent to the one introduced by Pachet et al. [45]. A weakness in these models is that their ability to find satisfying solutions diminishes quickly

as constraints are added or made more stringent. We quantitatively analyze this limitation in the applications section of this paper.

The constrained hidden Markov process (CHiMP) maintains the strengths of the CoMP and Factor graph models, in guaranteeing the generation of constraint-satisfying solutions, while significantly expanding the solution space to mitigate the limitations posed in these latter models by adding numerous or strict constraints.

5.3 Methods

The primary difference between the CoMP model and the CHiMP model is that the former derives from a Markov model whereas the latter derives from a *hidden* Markov model. As such, we will review these two models first and then look at how constraints are added to these models to form the CoMP and CHiMP models

As a running example we will consider the problem of generating a sequence with the following set of constraints (hereafter referred to as the set C):

1. The sequence must be four words in length
2. The first word rhymes with *red*
3. The last word is *red*

Furthermore the sequence of words must be generated according to probabilities derived from the following training corpus (note that for the sake of simplicity, and because it is irrelevant in the context of the constrained models, we purposely ignore end-of-sentence tags):

NNP	**RB**	**VBZ**	**NN**
Ted	*now*	*likes*	*green*

NNP	**VBZ**	**NN**
Mary	*likes*	*red*

NNP RB VBZ NN

Mary now loves red

NNP VBZ NNP RB

Fred sees Mary sometimes

The tokens NN, RB, VBZ, and NNP represent part-of-speech (POS) tags equating respectively to a noun, an adverb, a verb in 3rd-person singular present, and a proper noun singular. For purposes of equal comparison all models are required to incorporate both POS and words into their state spaces. Markov models do this by combining both POS and words into a single state space; hidden Markov models explicitly separate POS and words into hidden and observed state spaces respectively.

Markov Processes

A Markov model is defined as a triple $M = (S_M, \pi_M, T_M)$ where S_M defines a finite set of observed states or symbols; $\pi_M : S_M \to [0, 1]$ represents the initial probabilities for states; and $T_M : S_M \times S_M \to [0, 1]$ represents the transition probabilities between states. Each of π_M and T_M represent a distribution and should thus sum to 1.0.

For the training corpus above, $M = (S_M, \pi_M, T_M)$ where $S_M = \{$(**NNP**, *Ted*), (**RB**, *now*), (**VBZ**, *likes*), (**NN**, *green*), (**NNP**, *Mary*), (**NN**, *red*), (**VBZ**, *loves*), (**NNP**, *Fred*), (**VBZ**, *sees*), (**RB**, *sometimes*)$\}$, and π_M and T_M are defined as follows:

π_M	
(**NNP**,*Ted*)	1/4
(**NNP**,*Mary*)	2/4
(**NNP**,*Fred*)	1/4
else	0

T_M	(**NNP**, *Ted*)	(**RB**, *now*)	(**VBZ**, *likes*)	(**NN**, *green*)	(**NNP**, *Mary*)	(**NN**, *red*)	(**VBZ**, *loves*)	(**NNP**, *Fred*)	(**VBZ**, *sees*)	(**RB**, *sometimes*)
(**NNP**, *Ted*)	0	1	0	0	0	0	0	0	0	0
(**RB**, *now*)	0	0	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	0	0
(**VBZ**, *likes*)	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	0
(**NN**, *green*)	0	0	0	0	0	0	0	0	0	0
(**NNP**, *Mary*)	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0	0	0	0	$\frac{1}{3}$
(**NN**, *red*)	0	0	0	0	0	0	0	0	0	0
(**VBZ**, *loves*)	0	0	0	0	0	1	0	0	0	0
(**NNP**, *Fred*)	0	0	0	0	0	0	0	0	1	0
(**VBZ**, *sees*)	0	0	0	0	1	0	0	0	0	0
(**RB**, *sometimes*)	0	0	0	0	0	0	0	0	0	0

We generate a sequence $s = \{s_1, \ldots, s_n\}$ using M as follows:

1. Sample an initial state $s_1 \in S_M$ according to π_M
2. While sequence not finished,
   (a) Sample the next state $s_i \in S_M$ given s_{i-1} according to T_M

The sequence finishes either when an end-of-sequence token is sampled or when the sequence has reached some pre-determined length (as is the case in the CoMP and CHiMP models).

A Markov model M generates a sequence $s = \{s_1, \ldots, s_n\}$ of states from S_M with probability

$$P_M(s) = \pi_M(s_1) \prod_{i=1}^{n} T_M(s_{i-1}, s_i).$$

Trained on the example corpus, M generates the following sequences with the probabilities shown:

NNP	**VBZ**	**NN**		
Mary	*likes*	*green*		
1/2	$\times 1/3$	$\times 1/2$	=	1/12

NNP	**VBZ**	**NNP**	**RB**		
Fred	*sees*	*Mary*	*now*		
1/4	$\times 1$	$\times 1$	$\times 1/3$	=	1/12

Note that although the Markov process generates sequences according to probabilities derived from the training corpus, it generates sequences that fail to meet constraints as defined by C. The probability mass devoted to sequences which obey constraints in C is relatively small.

Hidden Markov Processes

A hidden Markov model is defined a five-tuple $H = (S_H, V_H, \pi_H, T_H, E_H)$ where S_H defines a finite set of (hidden) states; V_H is a finite set of observed states or symbols; $\pi_H : S_H \rightarrow [0, 1]$ represents the initial probabilities for hidden states; $T_H : S_H \times S_H \rightarrow [0, 1]$ represents the transition probabilities between hidden states; and $E : S_H \times V_H \rightarrow [0, 1]$ defines the probability of emitting a symbol given a particular hidden state. Each of π_H, T_H, and E_H represent a distribution and should thus sum to 1.0. For the training corpus above, $S_H = \{NN, RB, VBZ, NNP\}$, $V_H = \{$*Ted, now, likes, green, Mary, red, loves, Fred, sees, sometimes*$\}$, and π_H, T_H, and E_H defined as follows:

	π_H
NN	0
VBZ	0
RB	0
NNP	1

T_H	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	3/4	0	0	1/4
RB	0	1	0	0
NNP	0	2/5	3/5	0

E_H	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NNP	1/5	0	0	0	3/5	0	0	1/5	0	0
RB	0	2/3	0	0	0	0	0	0	0	1/3
VBZ	0	0	1/2	0	0	0	1/4	0	1/4	0
NN	0	0	0	1/3	0	2/3	0	0	0	0

To generate a sequence $s = \{(s_1, v_1), \ldots, (s_n, v_n)\}$, H follows the procedure:

1. Sample an initial hidden state $s_1 \in S_H$ according to π_H
2. Given s_1, sample a symbol $v_1 \in V_H$ according to E_H
3. While sequence not finished,
   (a) Sample the next hidden state $s_i \in S_H$ given s_{i-1} according to T_H
   (b) Sample the next symbol $v_i \in V_H$ given s_i according to E_H

The sequence finishes either when an end-of-sequence token is sampled or when the sequence has reached some pre-determined length.

Using H, a sequence $s = \{(s_1, v_1), \ldots, (s_n, v_n)\}$ is generated with probability

$$P_H(s) = \pi_H(s_1) E_H(s_1, v_1) \prod_{i=2}^{n} T_H(s_{i-1}, s_i) E_H(s_i, v_i)$$

Trained on the example corpus, H generates the following sequences with the probabilities shown:

1	$\times 2/5$	$\times 3/4$		
NNP	**VBZ**	**NN**		
Ted	*likes*	*green*		
$\times 1/5$	$\times 1/2$	$\times 1/3$	$=$	$1/100$

1	$\times 3/5$	$\times 1$	$\times 1/4$		
NNP	**RB**	**VBZ**	**NNP**		
Mary	*sometimes*	*loves*	*Fred*		
$\times 3/5$	$\times 1/3$	$\times 1/4$	$\times 1/5$	$=$	$3/2000$

Note again that although this model will generate sequences according to the correct probability distribution, the probability mass devoted to sequences which match our desired constraints is relatively small.

It is important to note that, trained on the same corpus, the solution space of H is a superset of the solution space for M (e.g., the example solutions shown for M can be generated by H, but the reverse is not true). This increased expressivity comes purely as a result of adding the hidden layer in H. In broad terms, the solution space of M is $O(|S_M|^n)$ (where n is the length of the sequence) whereas the solution space of H is $O(|S_H|^n \times |V_H|^n)$. Shifting from a Markov to a hidden Markov model increases the size of the solution space by an exponential factor, $|V_H|^n$. This enhanced expressivity of hidden Markov models with respect to non-hidden models is one of the keys that makes the CHiMP model more robust when adding constraints: an exponentially larger solution space significantly increases the chances of maintaining satisfying solutions when pruned by constraints.

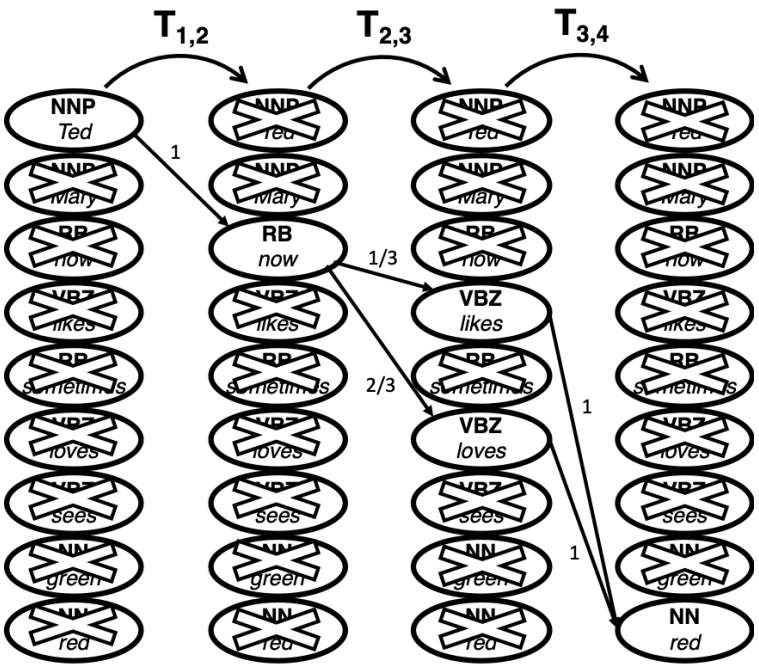


Figure 5.1: A constrained Markov process (CoMP) with constraints requiring the first token rhyme with *red* and the last token be *red*. Pruned states and updated transitions are the result of applying constraints and then enforcing arc-consistency.

Constrained Markov Processes

At a high-level, a CoMP model $\tilde{M}$, derived from a Markov model M and a set of constraints C, can be thought of as a Markov model that creates a copy $T_{i,j}$ of T for each pair of sequence positions (i, j) in the sequence to be generated and then modifies each $T_{i,j}$ to ensure that constraints in C applying to positions i and/or j are met (e.g., zero out probabilities in $T_{3,4}$ transitioning to any state $s_4 \neq red$). This is demonstrated in Fig. 5.1. The CoMP model $\tilde{M}$ applies arc-consistency and re-normalization to sample constraint-satisfying solutions s with probability $P_{\tilde{M}}(s)$ that differs from $P_M(s)$ by a constant factor. Because this model is a form of tree-structured CSP, arc-consistency can be enforced in a single pass to ensure probabilities sum to 1.0 and that relative probabilities for constraint-satisfying solutions are maintained (for the details on this algorithm and the algorithm for re-normalization see [42]). Because the transition probabilities can vary by position, the CoMP model is sometimes referred to as a *non-homogeneous Markov model*.

Given the training corpus and constraint set C, a trained CoMP model (as shown in Fig. 5.1) is able to generate only two solutions:

Sequence s	$P_M(s)$	$P_{\tilde{M}}(s)$
Ted now likes red.	1/16	1/3
Ted now loves red.	2/16	2/3

Note that $\tilde{M}$ only generates constrain-satisfying solutions and that these solutions are generated according to probabilities derived from our training corpus. The challenge, however, is that these are the only two solutions. In general, the number of satisfying solutions for $\tilde{M}$ is very few compared to what will be shown next for the CHiMP model.

Constrained Hidden Markov Processes

The CHiMP model follows a pattern similar to that of the CoMP model except that instead of combining a *non-hidden* Markov model with a set of constraints C, the CHiMP model combines a *hidden* Markov model with C. Given a hidden Markov model $H = (S_H, V_H, \pi_H, T_H, E_H)$ and a set of constraints C, a constrained hidden Markov process $\tilde{H}$ generates sequences of hidden and/or observed states that obey the constraints in C. $\tilde{H}$, like $\tilde{M}$, allows for probabilities to vary by position. The difference is that CHiMP replicates and modifies both T_H *and* E_H probabilities for each position (see Fig. 5.2). After applying constraints, arc-consistency is enforced to remove nodes that do not lead to a solution. As in $\tilde{M}$, arc-consistency measures can be enforced in $\tilde{H}$ in a single pass. All matrices are re-normalized to ensure that the distributions $P_{\tilde{H}}$ and P_H differ by only a constant factor for satisfying solutions.

Given the training corpus and our rhyming constraints, the CHiMP model is able to generate the following 12 solutions, each with their respective probabilities.

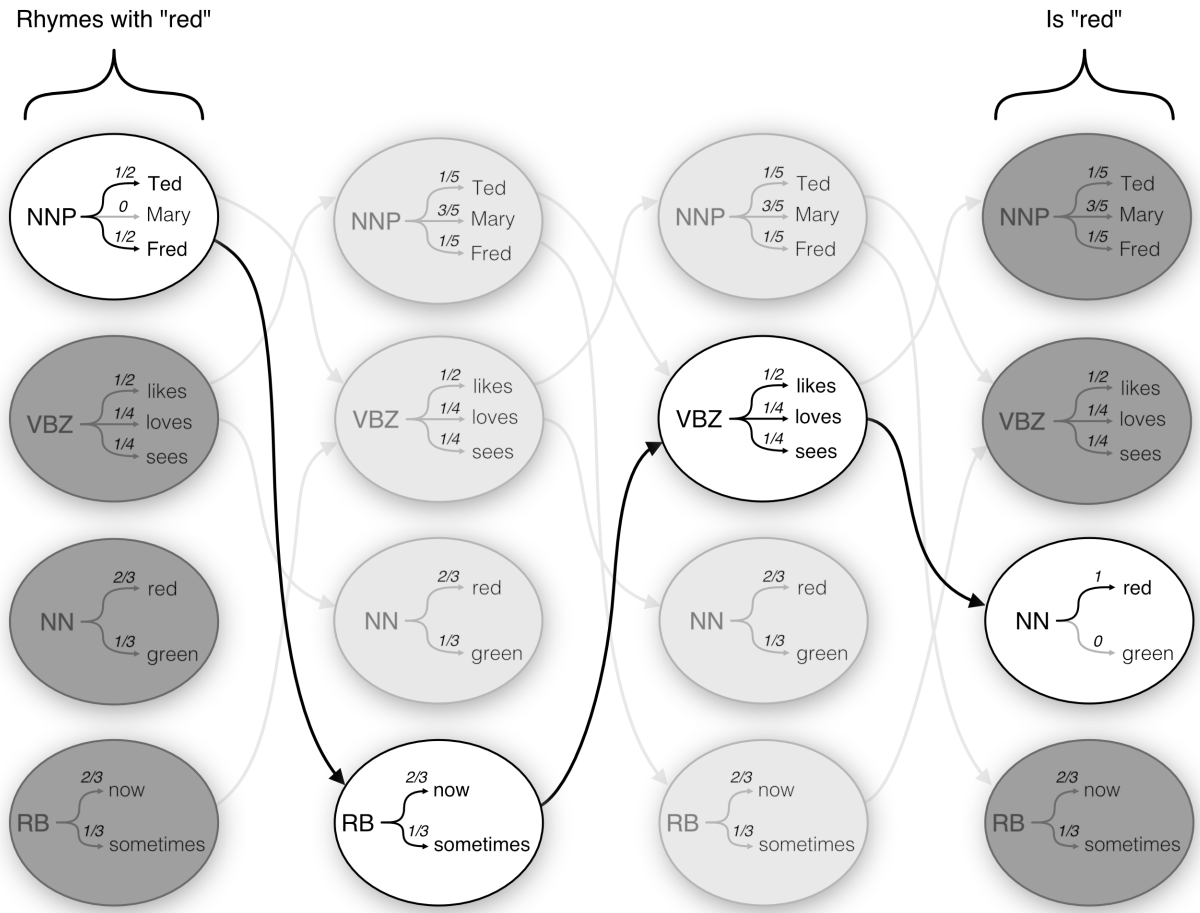


Figure 5.2: A high-level schematic of a constrained hidden Markov process (CHiMP) of length 4 constrained so that the last word is "red" and the first word rhymes with "red". Each column represents a position in the sequence to be generated. Each node represents a hidden state (i.e., part-of-speech) and a probability distribution for the observed states (i.e., words) that can be generated from that hidden state. By pruning observed states that are disallowed by constraints and then adjusting probabilities to maintain arc-consistency, the resulting model generates constraint-satisfying solutions with probability relative to the original probability distribution. Hidden states pruned directly from applying constraints are indicated by dark grey nodes and states pruned during arc-consistency are indicated by light grey nodes.

Solutions	Probabilities
Ted now likes red.	(4/200)
Ted now loves red.	(2/200)
Ted now sees red.	(2/200)
Ted sometimes likes red.	(2/200)
Ted sometimes loves red.	(1/200)
Ted sometimes sees red.	(1/200)
Fred now likes red.	(4/200)
Fred now loves red.	(2/200)
Fred now sees red.	(2/200)
Fred sometimes likes red.	(2/200)
Fred sometimes loves red.	(1/200)
Fred sometimes sees red.	(1/200)

By including hidden states in the constrained model $\tilde{H}$, the increased solution space results in the model being able to generate *significantly* more satisfying solutions than $\tilde{M}$. A second added benefit is afforded by the addition of hidden states: because constraints are applied on observed states and because there is no direct influence between observed states in H, there is an increased degree of separation between constraints. This helps to avoid the compounding effect of diminished state spaces resulting from individual constraints.

5.4 Applications

To demonstrate the improvements of the CHiMP model over the CoMP model, we compared the results of each model trained on the Corpus of Contemporary American English (COCA) [16] and provided the same set of constraints. In particular, we selected training sets from the 2012 fiction portion of COCA and constrained each model to only output sequences in

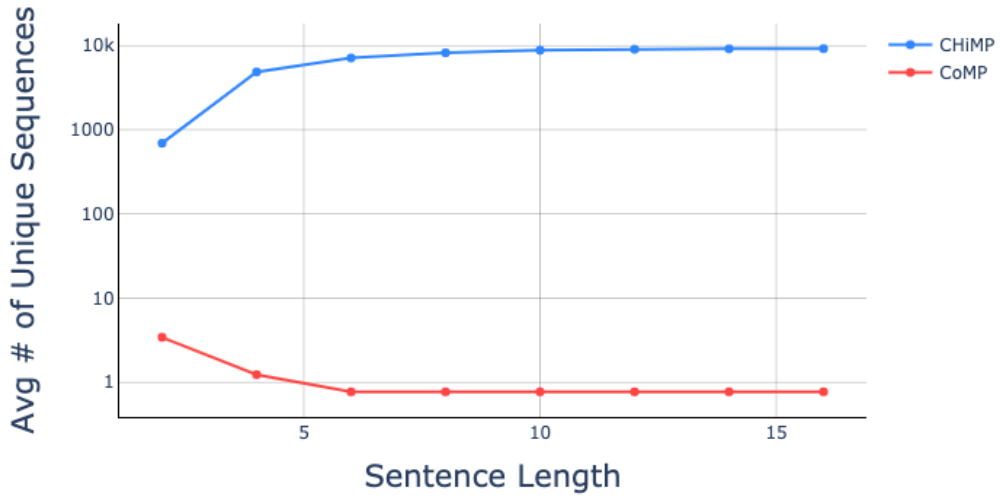


Figure 5.3: The effects of sequence length (and consequently number of constraints) on generalizability (i.e., number of unique sequences out of 10k sampled solutions) for a fixed random training set of 100 sentences. Each model is constrained such that words start with the same letter, and counts are averaged over 26 runs (a different letter constraint for each run). The added constraints from increasing sequence length have a compounding limiting effect in the CoMP model, whereas the abstraction of the CHiMP model serves to decouple constraints to avoid bottlenecks.

which the first letter of each word began with the same letter (e.g., a tongue-twister). We chose this problem because it represents a fairly general example of constrained sequence generation that is easily adapted to sequences of varying lengths. Results are averaged over 26 instances of the problem with each instance having constraints defined with a different letter of the English alphabet.

Our experiments were two-fold. First, we examined the number of sequences generated by each model as a function of sequence length. Since each word is constrained to start with a specified letter, as the sentence length increases, so does the number of constraints. This experiment compares how the models are affected by increasingly stringent constraints. Second, we examined the number of sequences generated by each model as a function of the size of the training corpus. Our motivation here is to compare how the models are affected by an increasingly restricted data set. In all cases, both models were trained on the same subset of data.

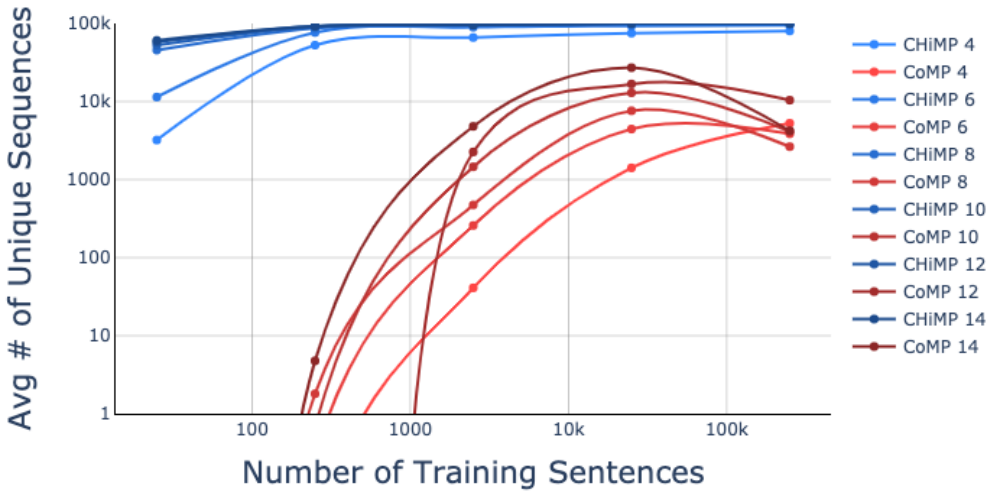


Figure 5.4: The effects of training corpus size on generalizability of the CHiMP (blue) and CoMP (red) models. Generalizability is measured as number of unique sequences out of 100k sampled solutions. Each model is constrained such that words start with the same letter, and counts are averaged over 26 runs (a different letter constraint for each run). Shades show the effects of varying the sequence length (and consequently the number of constraints) on generalizability. The CHiMP model consistently generates more unique satisfying solutions than the CoMP model and is relatively immune to the effects of training set size or number of constraints.

In Fig. 5.3, we see that as the sentence length increases and, by consequence of our problem, the constraints become more numerous, the CoMP model generate less and less unique sentences – even unable to generate sentences past a sentence length of 6 (for a fixed corpus size of 100 sentences). Each constraint added further tightens the restrictions put on possible solutions that CoMP can find. In contrast, CHiMP maintains expressivity (i.e., the ability to generate unique sentences) as sentence length grows and constraints are added.

In Fig. 5.4, CHiMP remains expressive even when trained on a very small training corpus (25 sentences). The trend of unique sentence averages increases as the training set increases until reaching the maximum number of sampled solutions (100K). CoMP's ability to generate novel and unique sentences is severely limited according to the size of its training set.

5.5 Conclusion

We have demonstrated through quantitative analysis that under increasingly stringent constraints and increasingly limited training data, constrained hidden Markov processes are an ideal solution for maintaining reasonable solution spaces. It remains to be seen what *qualitative* affect the CHiMP model has compared to the CoMP model in terms of the *coherence* of generated sequences. Regardless, insofar as quality is reflected in the ability of the model to add and respect more constraints, CHiMP maintains a significant advantage.

The diminished fear of adding constraints afforded by the CHiMP model is a significant asset that opens many new and exciting avenues for research in sequence generation. Where many problems are easily defined using constraints, the difficulty of generating structured sequences shifts from the inability of the model to find solutions to challenging the system designer to think of new and creative ways to derive and define constraints.

Chapter 6

Constrained Hidden Markov Processes for Sequence Generation

This paper will be submitted to AAAI 2023 in September 2022.

Abstract

There is growing interest in systems that generate natural and meaningful sequences. Many existing sequence generation models, including Markov and neural algorithms, capture local coherence but lack a mechanism for imposing structural constraints that are so often essential for the development of meaning. We propose a novel solution to this problem that combines hidden Markov models with constraints, allowing sequences that obey user-defined constraints. Compared to other constrained, probabilistic solutions, our Constrained Hidden Markov Process (CHiMP)[1] has significantly larger solution spaces, allowing the user to generate constrained sequences that have more numerous structural constraints. CHiMP also demonstrates high sequence quality comparable to the constrained non-hidden Markov model and the anticipation-RNN as assessed by a survey.

6.1 Introduction

Sequence generation is a task of interest in domains such as natural language, music [13], game content [27], etc. Markov processes can be implemented to run efficiently enough for complicated real-time tasks such as speech synthesis [62]. The efficiency of Markov processes comes in part from the Markov hypothesis of limited memory which is that the next state of

[1]Source code is available at *https://github.com/po-gl/ConstrainedHiddenMarkovModel*

a sequence depends only on the previous state. In terms of the probabilities of a sequence $x_1, \ldots, x_L$, the Markov hypothesis can be written as:

$$p(x_i|x_1, \ldots, x_{i-1}) = p(x_i|x_{i-1})$$

The Markov hypothesis becomes an obstacle when trying to enforce unary constraints upon a sequence. Fortunately, this obstacle has been overcome for non-hidden Markov processes by Pachet et al. [42] which we will refer to as the constrained Markov process (CoMP). Markov processes can be set up as constraint satisfaction problems (CSPs) in which states can be removed that violate constraints, arc-consistency can be enforced to guarantee solution paths, and transition probabilities normalized to ensure original probability distributions. The solution they present is extended here for a general solution to constraining *hidden* Markov processes.

To motivate why constraints are desired for sequences in the first place, consider the importance of structure to generated sequences. Music, for example, progresses sequentially; notes are followed by more notes. However, to elevate a musical phrase to be something interesting, higher features such as motifs need to be present in the sequence [1, 38]. Likewise coherent English sentences require proper subject-verb and noun-pronoun agreements, possibly between distant sequence positions. Quality generation of English sentences are important to any natural language application, but are increasingly in demand for applications such as virtual assistants or procedural content generation [32].

This paper aims to propose a solution to constraining a hidden Markov process – primarily based on extending the notion of the constrained Markov process from Pachet et al. [42] to hidden Markov models. The proposed solution is the constrained hidden Markov process (CHiMP). The methods described in this paper allow for a hidden Markov process, with a set of unary constraints, to be re-compiled into a new model which is a constrained hidden Markov process that is statistically equivalent to the original hidden Markov process. The remainder of the paper is structured as follows: relevant related works are discussed, a

problem statement and running sentence generation example is introduced, the construction of the model is described, model time complexities are justified, and the model is applied to a two sequence generation problems where quantitative and qualitative results are presented.

6.2 Related Works

A general method for creating a constrained non-hidden Markov model has been described by Pachet et al. where they apply the model to generating musical melodies [42]. Defining attributes of their approach include leveraging arc-consistency to address the zero-frequency problem common in random walk settings while maintaining the original probability distribution of the training set. Of course, the constrained non-hidden Markov model also respects the Markov hypothesis when sampling generated sequences. We compare the results of the proposed model with the model described by Pachet et al. [42].

The zero-frequency problem occurs in random walk settings when a token is chosen for which there is no satisfying continuation of the sequence [10]. While this problem can occur in any random walk setting, the problem is exacerbated when a model tries to satisfy user-defined constraints. Prior work has shown that constrained models often contend with diminishing solution space sizes when heavily constrained [21]. In dealing with the zero-frequency problem, proposed solutions can involve restarting the random walk [18]. For the non-hidden Markov model, Pachet et al. solved this problem by enforcing arc-consistency when training the model. This ensures any path from any given token will lead to a satisfying solution.

Factor graphs are functionally equivalent to the constrained non-hidden Markov model in offering a probabilistic model that can satisfy user-defined constraints [45]. Both of these models deal with diminishing solution space sizes for stringent constraint tasks.

Another model commonly applied to the task of sequence generation is the recurrent neural network (RNN). RNNs are different from factor graphs and constrained non-hidden Markov models in that they perform higher-dimensional interpolation of training instances

when making predictions [23]. This allows RNNs to exhibit more complex behavior through generated sequences. Due to the more complex behavior, RNNs are not affected by the zero-frequency problem as the probabilistic models are. However, RNNs typically do not make guarantees of satisfying user-defined constraints. An anticipation-RNN [24] aims to enforce unary constraints using a two stacked long short-term memory (LSTM) [23] architecture. The anticipation-RNN achieves success in constraining Bach chorales to user-defined unary constraints; however, the model does allow for generated sequences that do not satisfy constraints and thus still does not make guarantees. For the anticipation-RNN, difficult or "out-of-style" constraints can even have a poor constraint satisfaction rate (less than 50% of generated sequences satisfy constraints) [24].

6.3 Problem Statement

The problem of interest is to generate fixed-length sequences that adhere to control constraints. The proposed model aims to solve the problem of diminishing solution space sizes due to increasing Markov orders or constraint requirements by using a hidden Markov process H rather than non-hidden Markov model. It is also of interest that the model uses a random walk or stochastic process approach and that the constrained hidden Markov process $\widetilde{H}$ generate sequences on the same probability distribution as H.

As a running example we will consider the problem of generating a sequence with the following set of constraints (hereafter referred to as the set C):

1. The sequence must be four words in length
2. The first word rhymes with *red*
3. The last word is *red*.

Furthermore the sequence of words must be generated according to probabilities derived from the following training corpus (note that for the sake of simplicity, and because it is irrelevant in the context of the constrained models, we purposely ignore end-of-sentence tags):

NNP RB VBZ NN

Ted now likes green

NNP VBZ NN

Mary likes red

NNP RB VBZ NN

Mary now loves red

NNP VBZ NNP RB

Fred sees Mary sometimes

The tokens NN, RB, VBZ, and NNP represent part-of-speech (POS) tags equating respectively to a noun, an adverb, a verb in 3rd-person singular present, and a proper noun singular.

A hidden Markov process is defined as a 5-tuple $H = (S, O, \pi, M, E)$ where $S = \{s_1, \ldots, s_n\}$ is a set of hidden states; $O = \{o_1, \ldots, o_m\}$ is a set of observations (or, if the reader prefers, outputs); $\pi : S \to [0, 1]$ defines a set of initial state probabilities or prior probabilities; $M : S \times S \to [0, 1]$ defines a set of state transition probabilities; and $E : S \times O \to [0, 1]$ defines a set of emission probabilities. A generated sequence x of length L is defined by $x = (x_1, \ldots, x_L)$ where $x_i \in O$. We define X as the set of all generated sequences generated by H. Trained on the running example training corpus, a model H would have the following transition and emission probabilities:

π	
NN	0
VBZ	0
RB	0
NNP	1

M	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	3/4	0	0	1/4
RB	0	1	0	0
NNP	0	2/5	3/5	0

E	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NNP	1/5	0	0	0	3/5	0	0	1/5	0	0
RB	0	2/3	0	0	0	0	0	0	0	1/3
VBZ	0	0	1/2	0	0	0	1/4	0	1/4	0
NN	0	0	0	1/3	0	2/3	0	0	0	0

A constrained hidden Markov process is defined as a 5-tuple $\widetilde{H} = (S, O, \pi, \widetilde{M}, \widetilde{E})$ where $S = \{s_1, \ldots, s_n\}$ is a set of hidden states; $O = \{o_1, \ldots, o_m\}$ is a set of observations; $\pi : S \to [0, 1]$ defines a set of initial state probabilities or prior probabilities; $\widetilde{M}^{(i)} : S \times S \to [0, 1]$ defines each set of state transition probabilities for $1 < i < L$ and $\widetilde{M}^{(0)} = \pi$; and $\widetilde{E}^{(i)} : S \times O \to [0, 1]$ defines each set of emission probabilities for $1 \leq i < L$. The difference between $\widetilde{H}$ and H is that $\widetilde{H}$ has a series of L transition and emission matrices $\widetilde{M}^{(i)}$ and $\widetilde{E}^{(i)}$ respectively.

$\widetilde{H}$ is constructed with the use of unary constraints on the transition and emission probability matrices. We define the set of constraints on the hidden states at position i as ${C_M}^{(i)}$ for $1 \leq i < L$ and L is the length of the sequence to generate. We define the set of constraints on the observed states at position i as ${C_E}^{(i)}$ for $1 \leq i < L$. The set of constraints at a position specify what hidden or observed states are allowed, i.e., what states have a non-zero probability. In the running example, the constraint that the first word rhymes with red is specified by ${C_E}^{(0)}$ and the constraint that the last word is red is specified by ${C_E}^{(3)}$.

Using these constraints, we denote the set of solutions generated by $\widetilde{H}$ as X_C. Like CoMP, CHiMP should satisfy the following properties:

$$\text{(I) } p_{\widetilde{H}}(x) = 0 \text{ for } x \notin X_C$$

$$\text{(II) } p_{\widetilde{H}}(x) = \alpha \cdot p_H(x) \text{ for } x \in X_C$$

Property (I) states that $\widetilde{H}$ generates exactly the sequences $x \in X_C$, i.e., the model satisfies the control constraints. Property (II) states that the model $\widetilde{H}$ has the same probability distribution as its non-constrained variant H for sequences in X_C within some constant factor α We prove that CHiMP satisfies these properties in Section 6.4.5.

Section 6.4 will describe the construction of the constrained hidden Markov process $\widetilde{H}$ from an original model H such that $\widetilde{H}$ satisfies the two properties.

6.4 Construction of the Constrained Model $\widetilde{H}$

Like Pachet et al. [42], we construct the constrained model $\widetilde{H}$ by applying two transformations to our original hidden Markov model. The first transformation is to remove states that are directly or indirectly incompatible with the constraints. Directly incompatible states are states that constraints specifically forbid, and indirectly incompatible states are states that do not lead to a solution. Indirectly incompatible states are removed while enforcing arc-consistency. The second transformation is to normalize the model. After removing nodes, the matrices in the model become non-stochastic (matrix rows no longer add up to one). Normalization brings the matrices back to a stochastic state.

6.4.1 Extract Matrices from H

When a hidden Markov process is constructed, the end product is a transition matrix and an emission matrix. In constructing CHiMP, the process will be modeled by a series of L transition and emission matrices. To construct $\widetilde{H}$, a series of intermediate matrices $Z^{(0)}, \ldots, Z^{(L-1)}$ and

$B^{(0)}, \ldots, B^{(L-1)}$ are required to store the transition and emission probabilities respectively as states are removed when applying constraints and enforcing arc-consistency. The algorithm to initialize the intermediate matrices is as follows:

$$Z^{(0)} \leftarrow \pi$$
$$Z^{(i)} \leftarrow M, \forall i = 1, \ldots, L-1$$
$$B^{(i)} \leftarrow E, \forall i = 0, \ldots, L-1$$

meaning that the transition and emission probabilities of the original model H are copied $L-1$ and L times respectively into the intermediate matrices.

6.4.2 Applying Constraints

The first transformation in constructing $\widetilde{H}$ is to apply the control constraints ${C_M}^{(i)}$ and ${C_E}^{(i)}$ to the state spaces of the transition matrices $V_1, \ldots, V_L$ and the state spaces of emission matrices $U_1, \ldots, U_L$. For each constraint ${C_M}^{(i)}$ or ${C_E}^{(i)}$, we remove hidden states $a_s \in S$ or observed states $a_e \in O$ that are forbidden by the constraint from V_i or U_i respectively.

Now for each hidden state $a_s \in S$ that we removed from the domain V_i, we set the transitions to those words as zero in our list of transition matrices $Z^{(i)}$. Likewise we zero out probabilities in the emission probability matrices $B^{(i)}$ for each observed state $a_e \in O$ that is removed from the domain U_i. If all the emission probabilities for a given hidden state are all zero, then that hidden state is also removed from the domain V_i and zero out the transition to the hidden state. In our running example, the intermediate transition and emission matrices after initializing and applying constraints are as follows:

$Z^{(0)} = \pi$	
NN	0
VBZ	0
RB	0
NNP	1

$Z^{(1)}, Z^{(2)}, Z^{(3)}$	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	3/4	0	0	1/4
RB	0	1	0	0
NNP	0	2/5	3/5	0

$B^{(0)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	0	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0	0	0
NNP	1/5	0	0	0	0	0	0	1/5	0	0

$B^{(1)}, B^{(2)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	1/3	0	2/3	0	0	0	0
VBZ	0	0	1/2	0	0	0	1/4	0	1/4	0
RB	0	2/3	0	0	0	0	0	0	0	1/3
NNP	1/5	0	0	0	3/5	0	0	1/5	0	0

$B^{(3)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	2/3	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0	0	0
NNP	0	0	0	0	0	0	0	0	0	0

The constraints for the example only applying to the emission probabilities, i.e., C_M is empty while C_E has constraints for the first and last sequence position. Note that by zeroing some probabilities, the distribution no longer sums to 1.0 and will need to be normalized in order to satisfy property (II).

6.4.3 Enforcing Arc-consistency

Now that constraints have been applied and transition states removed, there is the possibility that a random walk will run into a dead-end (a state that has no transition possibilities from it). These dead-end states are removed through a process called *enforcing arc-consistency*. General algorithms for enforcing arc-consistency have been described [34], but enforcing arc-consistency gives the following:

$$\forall a \in V_i, \exists b \in V_{i+1} \text{ such that } {C_M}^{(i)}(a, b) = \ true$$

meaning that for all hidden states in the domain V_i there exists a hidden state in the next domain V_{i+1} such that the constraint at that position ${C_M}^{(i)}$ is satisfied. This process can be set up in the form of a tree-structured CSP and computed efficiently in a single pass without any backtracking. In practice, the process of enforcing arc-consistency zeros out transition probabilities to hidden states that lead to dead-end states.

The enforcement of arc-consistency in the constrained hidden Markov process solves the zero-frequency problem for the model. Given any or no constraints, enforcing arc-consistency ensures that an item leading to an unsatisfying continuation of the sequence cannot be chosen.

After the enforcement of arc-consistency, the intermediate transition and emission matrices for our sentence generation example will be as follows:

$Z^{(0)} = \pi$	
NN	0
VBZ	0
RB	0
NNP	1

$Z^{(1)}$	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	0	0	0	0
RB	0	0	0	0
NNP	0	0	3/5	0

$Z^{(2)}$	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	0	0	0	0
RB	0	1	0	0
NNP	0	0	0	0

$Z^{(3)}$	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	3/4	0	0	0
RB	0	0	0	0
NNP	0	0	0	0

$B^{(0)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	0	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0	0	0
NNP	1/5	0	0	0	0	0	0	1/5	0	0

$B^{(1)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	0	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0
RB	0	2/3	0	0	0	0	0	0	0	1/3
NNP	0	0	0	0	0	0	0	0	0	0

$B^{(2)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	0	0	0	0	0
VBZ	0	0	1/2	0	0	0	1/4	0	1/4	0
RB	0	0	0	0	0	0	0	0	0	0
NNP	0	0	0	0	0	0	0	0	0	0

$B^{(3)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	2/3	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0	0	0
NNP	0	0	0	0	0	0	0	0	0	0

6.4.4 Normalization

We now construct the final matrices $\widetilde{M}^{(0)}, \ldots, \widetilde{M}^{(L-1)}$ by normalizing the intermediate matrices $Z^{(0)}, \ldots, Z^{(L-1)}$. Normalizing the matrices individually would make the matrices stochastic; however, they would fail to satisfy property (II) and as such the constrained model $\widetilde{H}$ would not have the same probability distribution as the original model H.

The method to normalize while satisfying property (II) is based on the method to normalize a constrained non-hidden Markov process [42]. The process is another right-to-left sweep where the last matrix $Z^{(L-1)}$ is individually normalized and then the normalization is propagated back up to the prior matrix $Z^{(0)}$. Pachet et al describes the motivation for this process as propagating the perturbations in the matrices induced by individually normalizing matrices from right-to-left [42].

For a hidden Markov process, normalizing the transition probability matrices $Z^{(i)}$ is similar to normalizing matrices for a non-hidden Markov process; however, perturbations in the emission probabilities $B^{(i)}$ need to also be propagated. Perturbations in the emission probabilities are propagated through the matrices with the inclusion of a β factor. Emission probabilities themselves are calculated as $\widetilde{E}^{(i)}$.

The elements in the normalized matrices $\widetilde{M}^{(i)}$ and $\widetilde{E}^{(i)}$ are given by the recurrence relations below:

$$
\begin{aligned}
\widetilde{E}^{(i)}_{kx} &= \frac{B^{(i)}_{kx}}{\beta^{(i)}_k}, & \beta^{(i)}_k &= \sum_{x=1}^{|O|} B^{(i)}_{kx}, \quad 0 \leq i < L-1 \\
\widetilde{M}^{(L-1)}_{jk} &= \frac{\beta^{(L-1)}_k z^{(L-1)}_{jk}}{\alpha^{(L-1)}_j}, & \alpha^{(L-1)}_j &= \sum_{k=1}^{|S|} \beta^{(L-1)}_k z^{(L-1)}_{jk}, \quad 0 \leq i < L-1 \\
\widetilde{M}^{(i)}_{jk} &= \frac{\beta^{(i)}_k \alpha^{(i+1)}_k z^{(i)}_{jk}}{\alpha^{(i)}_j}, & \alpha^{(i)}_j &= \sum_{k=1}^{|S|} \beta^{(i)}_k \alpha^{(i+1)}_k z^{(i)}_{jk}, \quad 0 \leq i < L-1 \\
\widetilde{M}^{(0)}_{k} &= \frac{\beta^{(0)}_k \alpha^{(1)}_k z^{(0)}_{k}}{\alpha^{(0)}}, & \alpha^{(0)} &= \sum_{k=1}^{|S|} \beta^{(0)}_k \alpha^{(1)}_k z^{(0)}_{k}, \quad 0 \leq i < L-1
\end{aligned}
$$

The final matrices for the running example are utilized in Figure 6.1 and are as follows:

$\widetilde{M}^{(0)} = \pi$	
NN	0
VBZ	0
RB	0
NNP	1

$\widetilde{M}^{(1)}$	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	0	0	0	0
RB	0	0	0	0
NNP	0	0	1	0

$\widetilde{M}^{(2)}$	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	0	0	0	0
RB	0	1	0	0
NNP	0	0	0	0

$\widetilde{M}^{(3)}$	**NN**	**VBZ**	**RB**	**NNP**
NN	0	0	0	0
VBZ	1	0	0	0
RB	0	0	0	0
NNP	0	0	0	0

$\widetilde{E}^{(0)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	0	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0	0	0
NNP	1/2	0	0	0	0	0	0	1/2	0	0

$\widetilde{E}^{(1)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	0	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0
RB	0	2/3	0	0	0	0	0	0	0	1/3
NNP	0	0	0	0	0	0	0	0	0	0

$\widetilde{E}^{(2)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	0	0	0	0	0
VBZ	0	0	1/2	0	0	0	1/4	0	1/4	0
RB	0	0	0	0	0	0	0	0	0	0
NNP	0	0	0	0	0	0	0	0	0	0

$\widetilde{E}^{(3)}$	*Ted*	*now*	*likes*	*green*	*Mary*	*red*	*loves*	*Fred*	*sees*	*sometimes*
NN	0	0	0	0	0	1	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0	0	0
NNP	0	0	0	0	0	0	0	0	0	0

6.4.5 Proof of Properties (I) and (II)

We show through the following proof that the final normalized matrices satisfy property (I) and (II).

Proposition: The stochastic matrices $\widetilde{M}^{(i)}$ within the constrained hidden Markov process $\widetilde{H}$ satisfy property (I) and (II).

Proof. By construction, the matrices are stochastic, i.e., each row sums to 1. The probability that $\widetilde{H}$ will generate a given sequence $x = (x_1, \ldots, x_L)$ is:

$$\begin{aligned} p_{\widetilde{H}}(s) &= p_{\widetilde{H}}(s_1) \cdot p_{\widetilde{H}}(s_2|s_1) \cdot \ldots \cdot p_{\widetilde{H}}(s_L|s_{L-1}) \\ &= \widetilde{M}^{(0)}_{k_1} \cdot \widetilde{M}^{(1)}_{k_1,k_2} \cdot \ldots \cdot \widetilde{M}^{(L-1)}_{k_{L-1},k_L} \\ &= \frac{1}{\alpha^{(0)}} \cdot \beta^{(0)}_{k_1} \cdot z^{(0)}_{k_1} \cdot \beta^{(1)}_{k_1,k_2} \cdot z^{(1)}_{k_1,k_2} \cdot \ldots \cdot \beta^{(L-1)}_{k_{L-1},k_L} \cdot z^{(L-1)}_{k_{L-1},k_L} \end{aligned}$$

where k_i is the index of s_i in the alphabet S. Therefore, by the construction of the matrices $\widetilde{M}^{(i)}$:

$$\begin{aligned} &\text{(I) } p_{\widetilde{H}}(x) = 0 \text{ for } x \notin X_C \\ &\text{(II) } p_{\widetilde{H}}(x) = \frac{1}{a^{(0)}} \cdot p_H(x) \text{ for } x \in X_C \end{aligned}$$

□

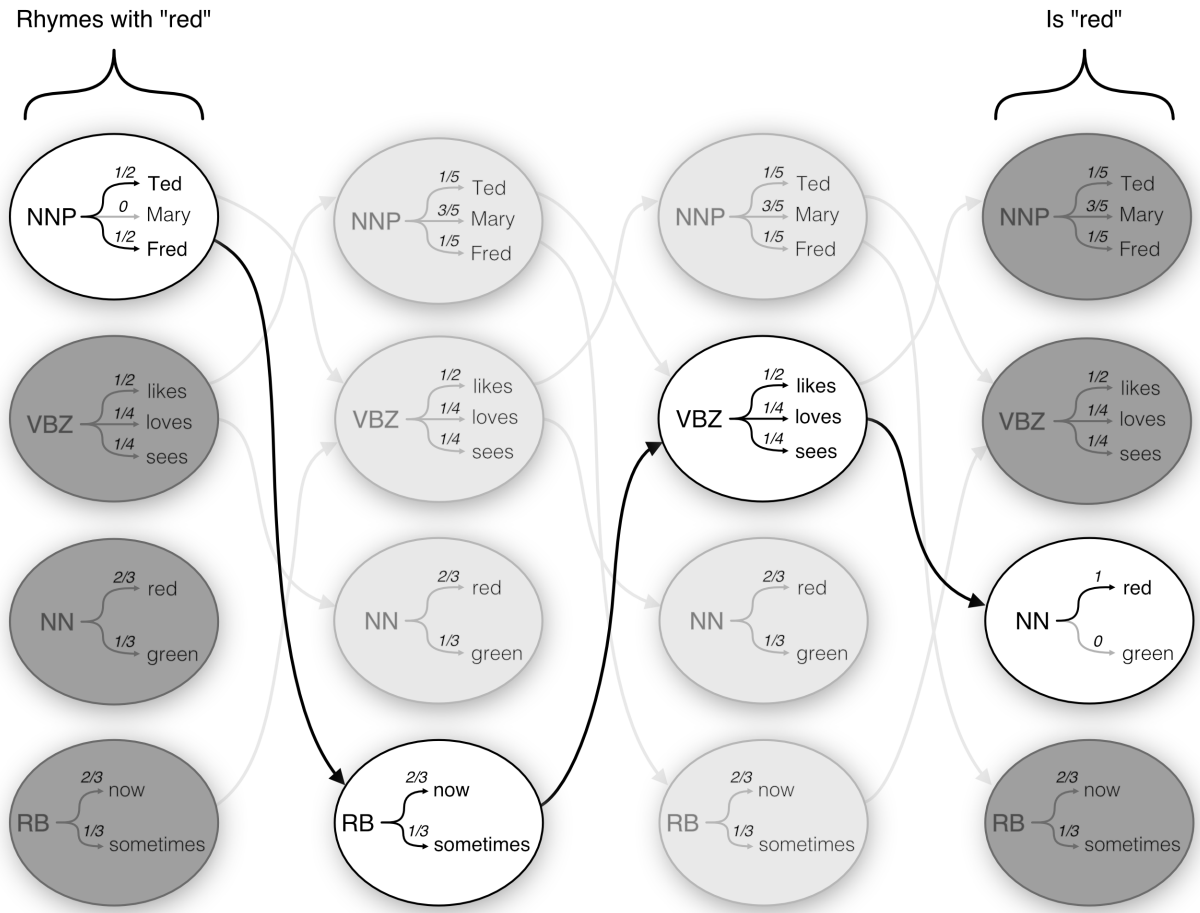


Figure 6.1: A high-level schematic of a constrained hidden Markov process (CHiMP) of length 4 constrained so that the last word is "red" and the first word rhymes with "red". Each column represents a position in the sequence to be generated. Each node represents a hidden state (i.e., part-of-speech) and a probability distribution for the observed states (i.e., words) that can be generated from that hidden state. By pruning observed states that are disallowed by constraints and then adjusting probabilities to maintain arc-consistency, the resulting model generates constraint-satisfying solutions with probability relative to the original probability distribution. Hidden states pruned directly from applying constraints are indicated by dark grey nodes and states pruned during arc-consistency are indicated by light grey nodes.

6.5 Time complexity

Constrained hidden Markov process training and sequence generation can be computed efficiently, enabling the possibility of using this model for real-time applications. The process of training, constraint satisfaction, arc-consistency enforcement, and normalization can be computed in a single pass. Sequence generation is a simple random walk that can be computed in linear time. The time complexities for the constrained hidden Markov process are as follows:

$$\text{Time to train model:} \quad O(t_{c_0} n + \sum_{i=1}^{L-1} (t_{c_i} n + n^2))$$

$$\text{Time to generate from model:} \quad O(Ln)$$

where t_{c_i} denotes the time it takes to process a state on the constraints ${C_M}^{(i)}$ and ${C_E}^{(i)}$ at sequence position i and n is the size of the alphabet with the assumption that $|S| = |O| = n$.

6.5.1 Proof

Proof. We will first prove the time to train the model which can be done in the following 4 steps:

1. construct the original hidden Markov model H
2. copy H to intermediate matrices $Z^{(i)}$
3. apply constraints and enforce arc-consistency
4. normalize matrices using the recurrence relations.

The first step of training the hidden Markov model on data can be done in $O(n^2)$ time; linear time to build the transition and emission matrices and $O(n^2)$ time to normalize the matrices. The second step can be ignored as far as time complexity since the original model can simply

be referenced as we build the matrices for $\widetilde{H}$. The third step will take $O(\sum_{i=1}^{L-1}(t_{c_i}n + n^2))$ time where for each position 1 to $L-1$, it will take $t_{c_i}n$ time to apply constraints in the matrix and n^2 time to enforce arc-consistency in a right-to-left single pass. At position 0, the model does not need to enforce arc-consistency further back, so we can add $t_{c_0}n$ time to apply constraints at position 0. Thus the third step will take $O(t_{c_0}n + \sum_{i=1}^{L-1}(t_{c_i}n + n^2))$. Finally, the fourth step will take $O(n^2)$ time to normalize each emission matrix $\widetilde{E}^{(i)}$ and another $O(n^2)$ time to normalize each transition matrix $\widetilde{M}^{(i)}$. The prior vector $\widetilde{M}^{(0)}$ can be processed in linear time $O(n)$. Therefore, the fourth step will take $n + \sum_{i=1}^{L-1}(2n^2)$ time or $O(n + \sum_{i=1}^{L-1}(n^2))$ time. Thus, the time to train the constrained hidden Markov model is as follows:

$$\begin{aligned} & O(n^2) + O(t_{c_0}n + \sum_{i=1}^{L-1}(t_{c_i}n + n^2)) + O(n + \sum_{i=1}^{L-1}(n^2)) \\ = \; & O(t_{c_0}n + \sum_{i=1}^{L-1}(t_{c_i}n + n^2)). \end{aligned}$$

Now the time to generate from the constrained hidden Markov model is a simple random walk on the matrices from left-to-right. Thus the time to generate will be the length of the sequence multiplied by number of possible transitions (alphabet size). Therefore, the time to generate is as follows:

$$O(Ln)$$

□

6.5.2 Experimental Validation

The experimental results shown in Figure 6.2 and Figure 6.3 support the theoretical time complexities. The model for the experimental results was trained using an arbitrary problem of numeric tokens that allowed for an increasing alphabet size. We can see that for an increasing alphabet size n that the time to train the model increases as a quadratic function

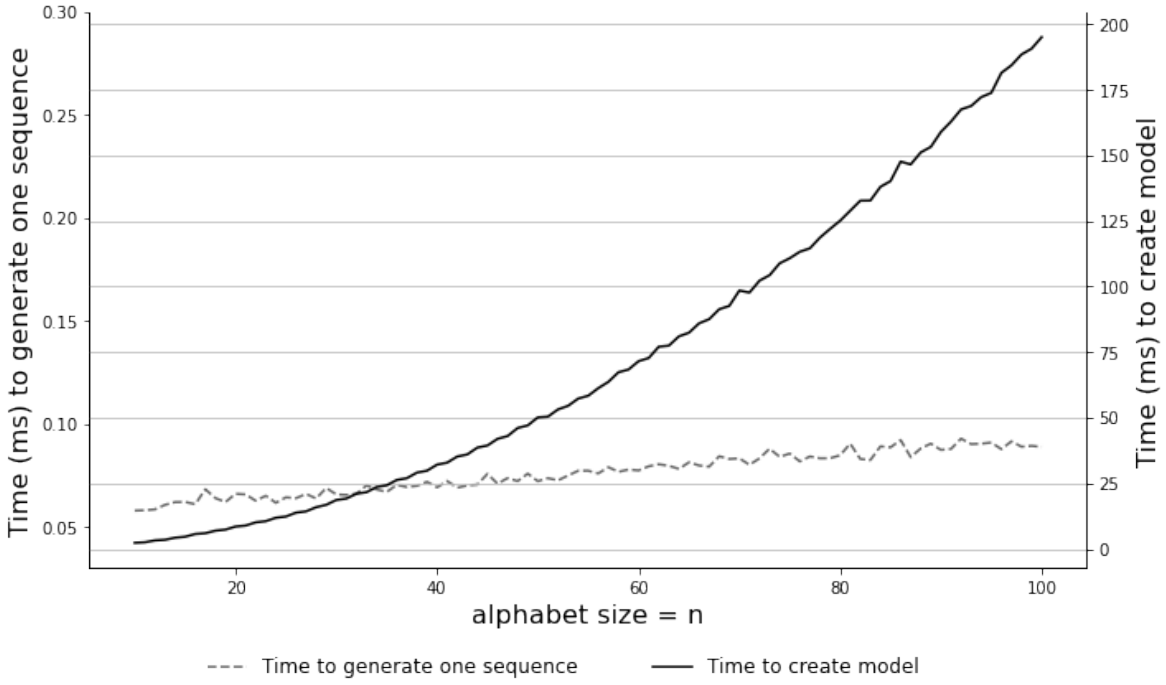


Figure 6.2: Experimental results that show the performance of CHiMP as a function of the size of the alphabet n. The solid line is the time to create (train) the model and shows the performance to be a quadratic function of n. The dashed line grows linearly with n.

of n. This corresponds to the n^2 term in the theoretical running time to train the model. In the same graph, we see that the time to generate one sequence grows linearly with n which corresponds to the theoretical running time to generate ($O(Ln)$). As we increase the sequence length L the experimental running time to train the model and generate sequences both grow linearly, which again corresponds to the theoretical running times proven in this paper.

Note that there is a simplified time complexity to train the model if we relax our upper bound slightly. If we ignore that the first position requires less processing time, we can relax the upper bound to get the following time to train the model:

$$O(L(t_{c_i}n + n^2)).$$

Ignoring constraints with non-trivial running times, the model's time complexity can be described as quadratic to train and linear to generate in terms of n. Thus the model is

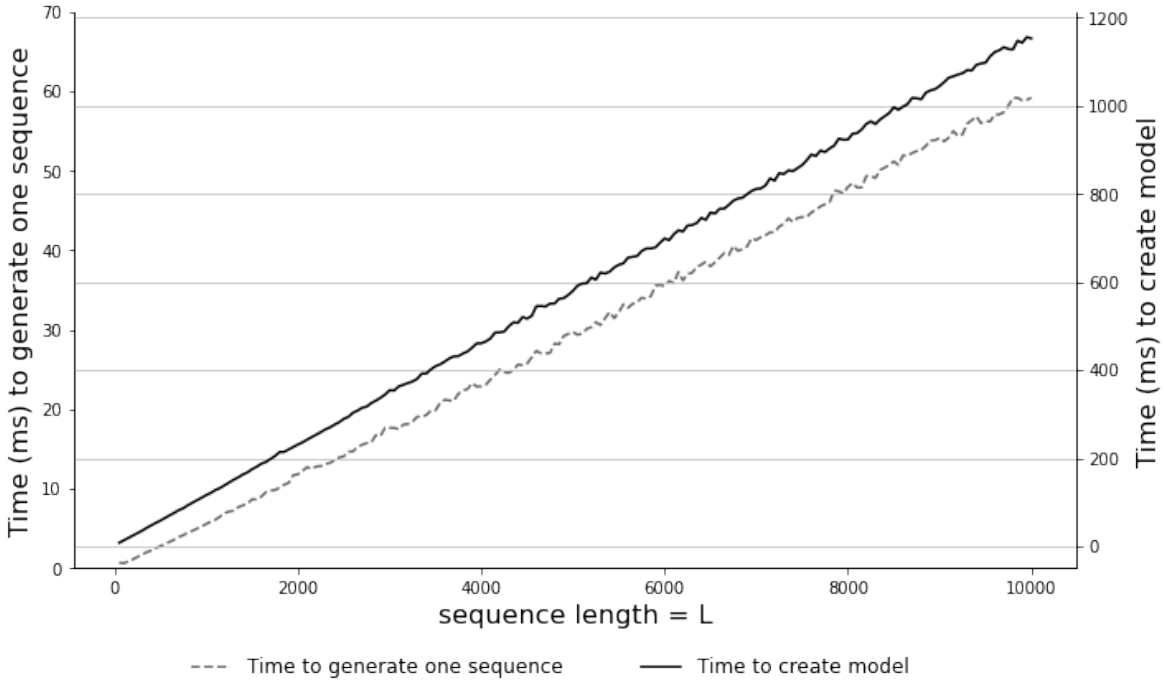


Figure 6.3: Experimental results that show the performance of CHiMP as a function of the length of the sequence L. The time to create (train) the model and generate a sequence both grow linearly with L.

efficient enough to be suitable for real-time applications such as assisting in real-time music composition. This is one of the constrained hidden Markov model's largest strengths over models such as neural networks which can require specialized hardware and hours of training time.

6.6 Results for Natural Language Sequence Generation

In demonstrating the improvements of CHiMP over CoMP, we devised a natural language task to generate sequences in which the first letter of each word begins with the same letter (i.e., a tongue-twister). The models were trained on works of fiction from the COCA dataset [16]. The task was chosen as it provides an even distribution of constraints while remaining general enough to adapt to varying lengths. Results are averaged over the 26 English alphabet letters. Note that the task constraints only affect the emission probabilities for CHiMP.

Figure 6.4: The effects of training corpus size on generalizability of the CHiMP (blue) and CoMP (red) models. Generalizability is measured as number of unique sequences out of 100k sampled solutions. Each model is constrained such that words start with the same letter, and counts are averaged over 26 runs (a different letter constraint for each run). Shades show the effects of varying the sequence length (and consequently the number of constraints) on generalizability. The CHiMP model consistently generates more unique satisfying solutions than the CoMP model and is relatively immune to the effects of training set size or number of constraints.

In Figure 6.4, we can see the effects of training corpus size on the number of unique sequences generated. CHiMP is able to generate thousands of unique sequences even the training corpus is small (25 sentences) whereas CoMP is unable to generate sequences with a small training corpus. The trend of unique sentence increases as the training set increases until reaching the maximum number of sampled solutions (100K). CoMP's ability to generate novel and unique sentences is severely limited according to the size of its training set.

In Figure 6.5, we see the effects of increasing the sequence length on the number of total solutions found by each model, i.e., the number of possible sequences that could be generated by each model. As the sentence length increases the constraints become more numerous due to the nature of our devised task. Since the transition matrices for CHiMP are not affected by the constraints in this task, we see an exponential increase in the total solutions for CHiMP whereas CoMP stagnates due to the restricting constraints. Attempts

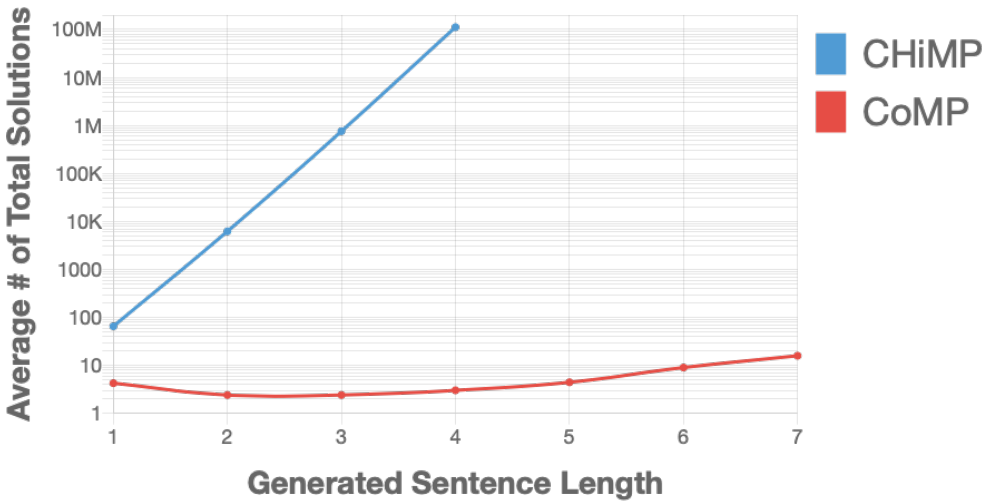


Figure 6.5: The effects of sequence length on the number of total solutions generated by each model with a fixed training set size of 300 sentences. Both models are constrained such that each word in a sequence starts with the same letter; counts of total solutions are averaged over 26 runs (each run using a different letter from the English alphabet). We see that as the sequence length increases, total solutions for the CHiMP model increases exponentially (given the logarithmic scale) whereas the CoMP model stagnates.

were made to apply the anticipation-RNN to this task; however, the model was unable to generate sequences that satisfied the constraints.

6.7 Results for Musical Sequence Generation

To demonstrate the quality of generated sequences, CHiMP was trained to generate four-part musical sequences in the style of J.S. Bach chorale harmonizations where the beginning and end of the sequence is constrained to match with the beginning and end of the first five measures of "Wer nur den lieben Gott läßt walten". An example of a generated sequence is shown in Figure 6.6. The generate sequences were then rated via a survey on how cohesive or natural-sounding they are. The same generative task is evaluated for CoMP and the anticipation-RNN. The dataset of chorale harmonizations used to train the models is available in the music21 python package [15]. In training the Markovian models CoMP and CHiMP, only pieces in a minor key are selected and then pieces are transposed into the key of C. In total, 174 chorales are used to train the two models. Having all the music in C minor is

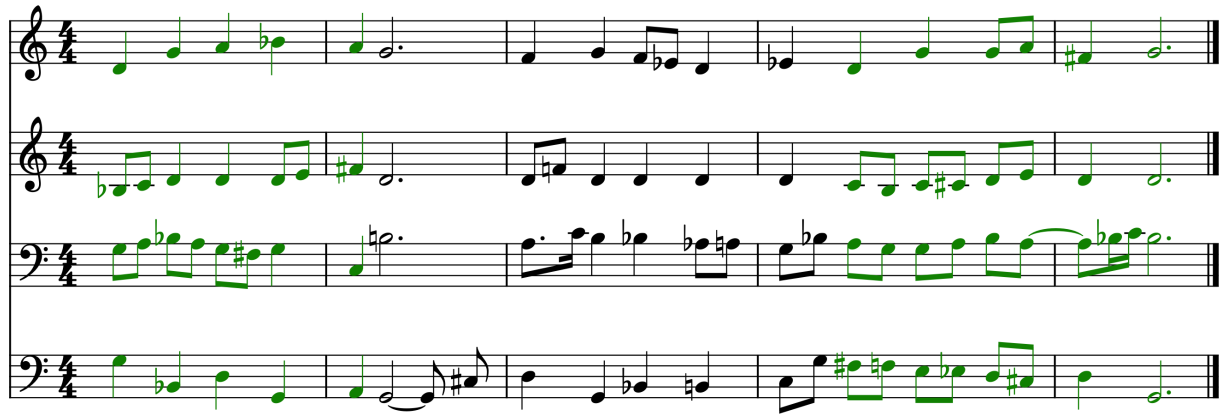


Figure 6.6: An example of a sequence generated by CHiMP with four note voices in the style of a Johann Sebastian Bach chorale. Green notes indicate the set of constraints used to generate the sequence. The constraints are to exactly match the beginning and end of the first five measures of "Wer nur den lieben Gott läßt walten".

intended to give the models a better chance of generating more cohesive sequences. If the dataset included chorales with differing keys, then generated sequences could allow transitions between notes in different keys which could create a jarring musical sequence.

From the 174 Bach pieces, the soprano, alto, tenor, and bass parts (or voices) are extracted into sequences. The polyphonic sequences are encoded using a method called *melodico-rhythmic* encoding [24]. In the encoding, time is quantized into 16^{th} note intervals where each beat in the music is divided into four equal parts. For each time slice, the token is a four-tuple where each position in the tuple is for each of the four voices. The name of the note is used if the note is played at that time slice, otherwise a new symbol "__" is used to indicate that the current note or rest is held.

For CHiMP, the soprano part of the time slice (melody) is used as the transition (hidden) states. The other three parts of the time slice (harmony) are used as the emissive (observed) states. A value from one to four is included in each emissive state that indicates the beat of the time slice. CoMP does not have separate transition and emissive states and transitions on states that include all four parts of the time slice as well as a beat indicator.

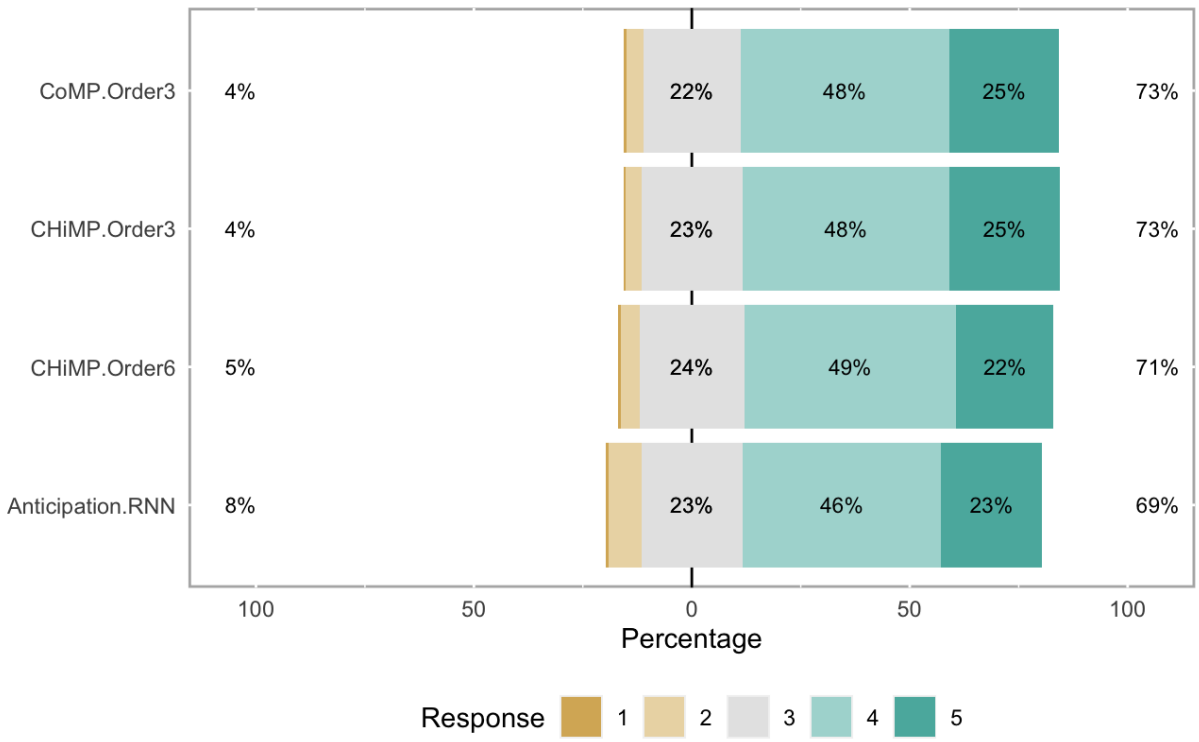


Figure 6.7: Survey results using Amazon's Mechanical Turks service yielded 2,400 responses (600 per model) and are visualized in a Likert chart. Participants were asked to rate a 12-second musical phrase on how cohesive (natural-sounding) the phrase sounds. The responses one to five correspond to: "very poor - Completely uncohesive music", "poor - mostly uncohesive music", "fair - equally cohesive and uncohesive music", "good - mostly cohesive music", and "excellent - completely cohesive music". For CHiMP trained with a Markov order of three, 73% of responses were four or five and only 4% of responses rated the music poorly on cohesiveness (one or two). The music phrases generated by CHiMP with Markov orders of three and six are rated similarly to CoMP whereas the phrases from the anticipation-RNN are rated slightly lower on cohesiveness.

The anticipation-RNN is trained in a similar way using the music21 dataset as outlined by Hadjeres and Nielsen [24]. In order to apply the anticipation-RNN to generate a musical sequence with four parts, some modifications to the implementation[2] were required.

An IRB approved survey was facilitated by Amazon's Mechanical Turk service where participants were asked to listen to the generated sequences and rate the cohesiveness of the musical phrase. The rating for cohesiveness ranges on an ordinal scale from one to five

[2]https://github.com/Ghadjeres/Anticipation-RNN

Table 6.1: Mann-Whitney U test p-values for cohesiveness survey result groups.

	CHiMP-Order3	CHiMP-Order6	Anticipation-RNN
CoMP-Order3	0.9774	0.2141	0.06142
CHiMP-Order3	-	0.2244	0.06369
CHiMP-Order6	-	-	0.4957

corresponding to the following labels: "very poor - Completely uncohesive music", "poor - mostly uncohesive music", "fair - equally cohesive and uncohesive music", "good - mostly cohesive music", and "excellent - completely cohesive music". For the survey, five sequences are generated for each model: CoMP with a Markov order of three, CHiMP with a Markov order of three, CHiMP with a Markov order of six, and the anticipation-RNN.

The survey results on sequence cohesiveness are shown in Figure 6.7. The 2,400 total ratings are overall positive with around 70% of responses being "good" or "excellent". Running a Mann–Whitney U test to test for statistical significance between the CoMP-Order3 and CHiMP-Order3 groups yields a high p-value of 0.9774 indicating insufficient evidence for the alternative hypothesis. Thus there is insufficient evidence that the difference in cohesiveness ratings for the two groups is statistically significant. Table 6.1 shows the Mann-Whitney U test p-values between the other model groups. There is insufficient evidence of a significant difference between any of the model groups.

The results indicate that the abstraction introduced in CHiMP does not significantly deteriorate the quality of generate sequences for the domain of music. The generated sequences and survey data are available online[3].

6.8 Conclusion

We have proposed an efficient method for constructing a hidden Markov process with control constraints. The constrained hidden Markov process improves on the non-hidden model by introducing a layer of abstraction that allows for greater solution space sizes and, in

[3]https://github.com/po-gl/MSThesis

turn, more complex constraints. Like with CoMP, CHiMP uses a random walk approach to generation and does not rely on search; thus the model is appropriate for real-time or interactive applications. Through arc-consistency, the model is able to solve the zero-frequency problem typical of random walk approaches. The model also retains the original probabilities of the training data within a constant factor.

In a natural language task, CHiMP remains expressive for restrictive constraints when compared to CoMP, demonstrating a high level of generalizability. In a music generation task, survey results provide evidence that CHiMP maintains sequence cohesiveness compared to CoMP and the anticipation-RNN. The qualitative results dispel concerns of deteriorating sequence quality caused by the abstraction in CHiMP. Thus CHiMP solves the problem of diminishing solution space sizes while maintaining sequence cohesiveness. Future work will investigate whether the model can maintain sequence cohesiveness for domains other than music.

Chapter 7

Conclusion

We have presented a novel method for constructing a model that is an improved solution to the problem of imposing structure onto generated sequences. The model has been demonstrated to be able to adhere to control constraints while maintaining a much larger solution space for restrictive constraints compared to previous solutions (e.g., CoMP).

In Chapter 3, we present a paper that demonstrates that constrained Markov processes (CoMPs) are well-suited to the task of constrained sequence generation. Chapter 4 formalizes the problem of diminishing solution space sizes that affect the CoMP model. CHiMP overcomes the problem of diminishing solution space sizes through its hidden states, introducing an element of abstraction. In the context of computational creativity, the larger solution space positions CHiMP as a model that systems can utilize to progress them further ahead on Ventura's spectrum of creative systems [64]. Chapter 5 initially describes CHiMP and presents results that demonstrate the enlarged solution space sizes. Finally, Chapter 6 fully describes CHiMP.

We have demonstrated that the proposed method for constructing CHiMP retains the desired properties of CoMP. In particular, the final model represents the same probability distribution as the original training set within a constant factor. The model uses a simple random walk approach to generating sequences and avoids the zero-frequency problem by enforcing arc-consistency when constructing the model. In the domain of music sequence generation, qualitative results suggest that sequence quality, as measured by sequence cohesiveness, does not deteriorate with CHiMP. The training and generating time complexities

of CHiMP are proven to be efficient, making the model suitable for real-time, interactive applications.

While CHiMP maintains its sequence cohesiveness in the domain of music sequence generation, there is concern that the model does not maintain cohesiveness in other domains. For instance, natural language poses a problem where a given part-of-speech (hidden state) can be tens of thousands of different words (observed state), which leads to generated sentences full of unrelated words. Future work may solve this problem with extended parts-of-speech. Future work will also investigate the efficaciousness of CHiMP for other domains.

To summarize the contributions of CHiMP presented in this thesis, we leave the reader with a metaphor where generating constrained sequences is akin to chiseling a marble block to expose the sculpture already there. Chiseling off material from the marble block is like adding constraints to the model. Adding strict or numerous constraints might chisel too much material and destroy the sculpture, i.e., the solution space size becomes too small to use. CHiMP, through its abstraction introduced by the hidden states, makes the marble block larger. With CHiMP, we can chisel away more material (adding strict or numerous constraints) and still have a sculpture (produce a usable solution space).

References

[1] Hussain-Abdulah Arjmand, Jesper Hohagen, Bryan Paton, and Nikki S. Rickard. Emotional Responses to Music: Shifts in Frontal Brain Asymmetry Mark Periods of Musical Change. *Frontiers in Psychology*, 8(DEC):2044, dec 2017. ISSN 1664-1078. doi: 10.3389/fpsyg.2017.02044. URL `http://journal.frontiersin.org/article/10.3389/fpsyg.2017.02044/full`.

[2] Gabriele Barbieri, François Pachet, Pierre Roy, and Mirko Degli Esposti. Markov constraints for generating lyrics with style. In *Ecai*, volume 242, pages 115–120, 2012.

[3] Margaret A. Boden. *The Creative Mind: Myths and Mechanisms, Second Edition*. Routledge, 2003. ISBN 0203508521. doi: 10.4324/9780203508527.

[4] Paul Bodily, Benjamin Bay, and Dan Ventura. Computational creativity via human-level concept learning. In *Proceedings of the Eighth International Conference on Computational Creativity*, pages 57–64, 2017.

[5] Paul M. Bodily, Porter Glines, and Brandon Biggs. "She Offered No Argument": Constrained Probabilistic Modeling for Mnemonic Device Generation. In *Proceedings of the 10th International Conference on Computational Creativity*, pages 81–88, Charlotte, North Carolina, 2019. Association for Computational Creativity.

[6] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[7] F. P. Brooks, A. L. Hopkins, P. G. Neumann, and W. V. Wright. An Experiment in Musical Composition. *IRE Transactions on Electronic Computers*, EC-6(3):175–182, 1957. ISSN 03679950. doi: 10.1109/TEC.1957.5222016.

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[9] Lee Cheatley, Wendy Moncur, and Alison Pease. Opportunities for computational creativity in a therapeutic context. In *10th International Conference on Computational Creativity*, pages 341–345. Association for Computational Creativity, 2019.

[10] John G Cleary and William J Teahan. Experiments on the zero frequency problem. In *Proc. Data Compression Conference*, volume 480, 1995.

[11] Simon Colton. Creativity Versus the Perception of Creativity in Computational Systems. *Proceedings of the AAAI Spring Symposium on Creative Systems*, 2008.

[12] Simon Colton and Geraint A. Wiggins. Computational creativity: The final frontier? In *Proceedings of the Twentieth European Conference on Artificial Intelligence*, pages 21–26. IOS Press, 2012. ISBN 9781614990970. doi: 10.3233/978-1-61499-098-7-21. URL `http://www.idi.ntnu.no/~agnar/Documents/Colton_Wiggins12.pdf`.

[13] Darrell Conklin. Music generation from statistical models. In *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pages 30–35. Citeseer, 2003.

[14] Mihály Csíkszentmihályi. *Flow and the Psychology of Discovery and Invention*. Harper Perennial, 1996. ISBN 0060928204. doi: 10.1037/e586602011-001.

[15] Michael Scott Cuthbert and Christopher Ariza. Music21: A toolkit for computer-aided musicology and symbolic music data. In J. Stephen Downie and Remco C. Veltkamp, editors, *ISMIR*, pages 637–642. International Society for Music Information Retrieval, 2010. ISBN 978-90-393-53813. URL `http://dblp.uni-trier.de/db/conf/ismir/ismir2010.html#CuthbertA10`.

[16] Mark Davies. The 385+ million word Corpus of Contemporary American English (1990–2008+): Design, architecture, and linguistic insights. *International Journal of Corpus Linguistics*, 14(2):159–190, 2009.

[17] Nina Dethlefs and Heriberto Cuayáhuitl. Hierarchical reinforcement learning and hidden markov models for task-oriented natural language generation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 654–659, 2011.

[18] Shlomo Dubnov, Gerard Assayag, Olivier Lartillot, and Gill Bejerano. Using machine-learning methods for musical style modeling. *Computer*, 36(10):73–80, 2003.

[19] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103: 48, 2002.

[20] Brendan J Frey, Frank R Kschischang, Hans-Andrea Loeliger, and Niclas Wiberg. Factor graphs and algorithms. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 35, pages 666–680. Citeseer, 1997.

[21] Porter Glines, Brandon Biggs, and Paul M. Bodily. A leap of creativity: From systems that generalize to systems that filter. In Kazjon Grace, Michael Cook, Dan Ventura, and Mary Lou Maher, editors, *Proceedings of the 11th International Conference on Computational Creativity*, pages 297–302. Association for Computational Creativity, 2020.

[22] Porter Glines, Brandon Biggs, and Paul Bodily. Probabilistic generation of sequences under constraints. In *Proceedings of the First Intermountain Engineering, Technology, and Computing Conference*, in press.

[23] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[24] Gaëtan Hadjeres and Frank Nielsen. Anticipation-rnn: Enforcing unary constraints in sequence generation, with application to interactive music generation. *Neural Computing and Applications*, pages 1–11, 2018.

[25] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *International Conference on Machine Learning*, pages 1362–1371. PMLR, 2017.

[26] Todd R Haskell and Maryellen C MacDonald. Constituent structure and linear order in language production: evidence from subject-verb agreement. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 31(5):891, 2005.

[27] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1–22, 2013.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[29] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[30] Anna Jordanous. Four PPPPerspectives on computational creativity in theory and in practice. *Connection Science*, 28(2):194–216, 2016. doi: 10.1080/09540091.2016.1151860.

[31] Leonard Koren. *Which "aesthetics" do you mean? : Ten definitions*. Imperfect Publishing, Point Reyes, California, 2010.

[32] Leo Leppänen, Myriam Munezero, Mark Granroth-Wilding, and Hannu Toivonen. Data-driven news generation for automated journalism. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 188–197, 2017.

[33] H-A Loeliger. An introduction to factor graphs. *IEEE Signal Processing Magazine*, 21 (1):28–41, 2004.

[34] Alan K Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1): 99–118, 1977.

[35] Richard G Morris, Scott H Burton, Paul M Bodily, and Dan Ventura. Soup Over Bean of Pure Joy : Culinary ruminations of an artificial chef. In *Proceedings of the Third International Conference on Computational Creativity*, pages 119–125, 2012. ISBN 9781905254668. URL `http://computationalcreativity.net/iccc2012/wp-content/uploads/2012/05/119-Morris.pdf`.

[36] Aran Nayebi and Matt Vitelli. GRUV: Algorithmic Music Generation using Recurrent Neural Networks. *Deep Learning for Natural Language Processing*, 2015.

[37] David Norton, Derrall Heath, and Dan Ventura. Finding creativity in an artificial artist. *Journal of Creative Behavior*, 2013. ISSN 00220175. doi: 10.1002/jocb.27.

[38] Joseph C. Nunes, Andrea Ordanini, and Francesca Valsesia. The power of repetition: Repetitive lyrics in a song increase processing fluency and drive market success. *Journal of Consumer Psychology*, 25(2):187–199, 2014. ISSN 10577408. doi: 10.1016/j.jcps.2014.12.004.

[39] Kyo-Joong Oh, Dongkun Lee, Byungsoo Ko, and Ho-Jin Choi. A chatbot for psychiatric counseling in mental healthcare service based on emotional dialogue analysis and sentence generation. In *2017 18th IEEE International Conference on Mobile Data Management (MDM)*, pages 371–375. IEEE, 2017.

[40] Balder Onarheim and Michael Mose Biskjaer. Balancing constraints and the sweet spot as coming topics for creativity research. In *Creativity in design: Understanding, capturing, supporting*. APA, 2017. URL `http://orbit.dtu.dk/files/103339029/Balancing_Constraints_and_the_Sweet_Spot.pdf`.

[41] François Pachet. The continuator: Musical interaction with style. *Journal of New Music Research*, 32(3):333–341, 2003. doi: 10.1076/jnmr.32.3.333.16861. URL `https://www.tandfonline.com/doi/abs/10.1076/jnmr.32.3.333.16861`.

[42] François Pachet, Pierre Roy, and Gabriele Barbieri. Finite-length markov processes with constraints. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[43] Alexandre Papadopoulos and Pierre Roy. Avoiding plagiarism in Markov sequence generation. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2731–2737, 2014. URL `www.aaai.org`.

[44] Alexandre Papadopoulos, Pierre Roy, and François Pachet. Avoiding plagiarism in markov sequence generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, pages 2731–2737, 2014.

[45] Alexandre Papadopoulos, François Pachet, Pierre Roy, and Jason Sakellariou. Exact sampling for regular and markov constraints with belief propagation. In *International Conference on Principles and Practice of Constraint Programming*, pages 341–350. Springer, 2015.

[46] Barak A Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[47] Alison Pease, Markus Guhe, and Alan Smaill. Some aspects of analogical reasoning in mathematical creativity. In *Proceedings of the First International Conference on Computational Creativity*, pages 60–64, 2010.

[48] Guillaume Perez and Jean-Charles Régin. Mdds: Sampling and probability constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 226–242. Springer, 2017.

[49] Rafael Pérez y Pérez and Mike Sharples. Three computer-based models of storytelling: BRUTUS, MINSTREL and MEXICA. *Knowledge-Based Systems*, 2004. ISSN 09507051. doi: 10.1016/S0950-7051(03)00048-0.

[50] Julie Porteous and Marc Cavazza. Controlling narrative generation with planning trajectories: the role of constraints. In *Joint International Conference on Interactive Digital Storytelling*, pages 234–245. Springer, 2009.

[51] L. R. Rabiner and B. H. Juang. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, 3(1):4–16, 1986. ISSN 07407467. doi: 10.1109/MASSP.1986.1165342.

[52] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[53] Graeme Ritchie. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines*, 17(1):67–99, 2007. ISSN 09246495. doi: 10.1007/s11023-007-9066-2. URL `https://link.springer.com/content/pdf/10.1007%2Fs11023-007-9066-2.pdf`.

[54] Stéphane Rivaud and François Pachet. Sampling Markov models under constraints: Complexity results for binary equalities and grammar membership. *arXiv preprint*, 2017. URL `https://arxiv.org/pdf/1711.10436.pdf`.

[55] Pierre Roy and François Pachet. Enforcing meter in finite-length markov sequences. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[56] Pierre Roy, Guillaume Perez, Jean-Charles Régin, Alexandre Papadopoulos, François Pachet, and Marco Marchini. Enforcing Structure on Temporal Sequences: The Allen Constraint. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 786–801. Springer, 2016. doi: 10.1007/978-3-319-44953-1{_}49. URL `https://www.researchgate.net/publication/304490456`.

[57] Warren S. Sarle. Neural networks and statistical models, 1994.

[58] Rob Saunders and John S Gero. The Digital Clockwork Muse: A Computational Model of Aesthetic Evolution. In *Proceedings of the Artificial Intelligence and Simulation of Behavior Convention*, pages 12–21, 2001. URL `https://pdfs.semanticscholar.org/420b/9355610e47a21398024ff3b423d66a002717.pdf`.

[59] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[60] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in neural information processing systems*, pages 1601–1608, 2009.

[61] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[62] Keiichi Tokuda, Yoshihiko Nankaku, Tomoki Toda, Heiga Zen, Junichi Yamagishi, and Keiichiro Oura. Speech synthesis based on hidden Markov models. *Proceedings of the IEEE*, 101(5):1234–1252, 2013. ISSN 00189219. doi: 10.1109/JPROC.2013.2251852.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[64] Dan Ventura. Mere Generation: Essential Barometer or Dated Concept. *Proceedings of the Seventh International Conference on Computational Creativity, ICCC*, pages 22–29, 2016. URL `https://pdfs.semanticscholar.org/51eb/ae710bf2c884b6c0b1479085604554fc9da4.pdf`.

[65] Dan Ventura. How to Build a CC System. In *Proceedings of the Eighth International Conference on Computational Creativity*, pages 253–260, 2017. URL `http://computationalcreativity.net/iccc2017/ICCC_17_accepted_submissions/ICCC-17_paper_20.pdf`.

[66] Geraint A. Wiggins. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7):449–458, 2006. doi: 10.1016/j.knosys.2006.04.009.

[67] Byung-Jun Yoon. Hidden markov models and their applications in biological sequence analysis. *Current genomics*, 10(6):402–415, 2009.