Alma Mater Studiorum
Università di Bologna

Dottorato di Ricerca in
MATEMATICA

Ciclo XXV

Settore Concorsuale: 01/A4
Settore Scientifico disciplinare: MAT/07

MARKOV CONSTRAINTS FOR GENERATING TEXTS WITH STYLE

Tesi di Dottorato presentata da: Gabriele Barbieri

Coordinatore Dottorato:
Chiar.mo Prof.
Alberto Parmeggiani

Relatore:
Chiar.mo Prof.
Mirko Degli Esposti

Esame Finale anno 2013

Contents

List of Figures

List of Tables

Introduction

This thesis address the issue of generating texts in the style of an existing author, that also satisfy structural constraints imposed by the genre of the text. The style of a text is an important factor that determines its quality, its legibility and its identity. The idea of generating new texts in the style of an existing author has been popular since the invention of Markov processes, that have shown to capture, at least in a first approximation, elements of the style of a corpus [20, 22]. Markov processes represent faithfully local properties of sequences, at varying orders, which makes them well-suited for such a task.

However, a text is not just a Markovian sequence, and, notwithstanding the issue of meaning, it has been shown that texts also exhibit statistical long-range correlations [2, 1]. For instance, poems or song lyrics often have rhymes or metric constraints, that are not always defined as local properties. They induce long-range dependencies that violate the hypothesis of short-term memory of Markov processes and demand long distance modeling. As a consequence, most approaches in automatic generation of stylistically imitative texts based on Markov models fail to capture higher-level properties of texts, which limit their use for practical applications, such as machine translation [40] or automatic summarization [5].

This thesis shows that the framework of Constrained Markov Processes (CMP) allows to precisely generate texts that are consistent with a corpus, while being controllable in terms of rhymes and meter, a result that no other technique, to my knowledge, could achieve to date. Constrained Markov processes addresses the issue of imposing constraints to a Markov process and generating structured Markov sequences by reformulating Markov Processes as constraint satisfaction problems. This thesis will show that lyrics generation can be formulated and solved in the CMP framework by expressing style as a set of *Markov constraints*, and properties of grammaticality, rhymes, meter and even to some extent, semantics, as constraints in that framework.

The first chapter of the thesis discusses related problems. The first part discuss what is the style of a text from a general point of view. Then it explains how the style of a text can modeled as a Markov process learned from the text itself. The second part introduces the field of natural language generation, starting from a brief history of natural language generation. The final part of the chapter covers the state of the art literature about two subfields of natural language generation, that are related to the goal of the thesis: generation of text with style and poetry generation.

The second chapter describes the framework of Constrained Markov Processes. The chapter begins by introducing Markov process and constraint satisfaction. Each technique is formally described, showing its benefits and drawbacks when used in the framework of natural language generation. Markov processes allow for easy and fast generation of texts with style, but without control on the structure of the generated texts. On the other hand constraint satisfaction techniques can easily used to generate texts that satisfy structural constraints, but at the cost of loosing control on the style of the generated texts. The final part of this chapter presents Constrained Markov processes. showing than they can precisely generate texts that are stylistically consistent with a corpus, while being controllable in terms of structural constraints.

The fourth chapter explains how to use Constrained Markov processes for poetry generation. More precisely it shows that poetry generation can be formulated and solved in the constrained Markov processes framework by expressing style as a set of Markov constraints, and properties of grammat-

icality, rhymes, meter and even to some extent, semantics, as control constraints.
The fifth chapter shows in details how the approach described in the previous chapters can be used for the semi-automatic generation of lyrics in the style of the songwriter Bob Dylan that has the same structure as an existing song (Yesterday by The Beatles). The same generation process is used to create lyrics in the style of more than 60 other authors. The results are listed at the end of this chapter.
Chapter 6 presents an empirical evaluation of the control constraints presented in chapter 3. The approach presented in the thesis is evaluated by asking humans to rate texts generated by constrained Markov processes against texts generated using pure Markov and pure constraint-based approaches. This evaluation shows that constrained Markov processes generate better texts in terms of syntactic correctness and semantic relatedness.
The last chapter of the thesis describes in details the implementation of an augmented text editor, called Perec. This editor is intended to improve creativity, by helping the user to write lyrics and poetry, exploiting the techniques presented so far. The user can specify the metric and the rhyme structure of the lyrics, and the concepts to talk about. Perec will then suggest new verses that satisfy all the control constraints imposed by the user. Finally the user can either accept or discard the proposed verses. The user can eventually modify the suggestion or incorporate it as apart of human created lyrics.

Chapter 1

Generating texts with style

1.1 The style of a text

Imagine a scientific paper wrote as a song by Bob Dylan, or a comedy wrote using the dry lexicon of a software's documentation. Nevertheless the results can be somehow interesting, these examples show how the style of a text is an important factor that determines its quality, its legibility and its identity.
But what is style? Style can be loosely defined as the way something is done. People can intuitively recognize if the styles of two novels, stories, or lyrics are similar or different. Everyone tends to classify people or things according to their different style, and style is one of the key aspect to appreciate or dislike an artwork. Style is a way to communicate our impressions and feelings when talking about an author. Different kinds of style can be connected to categories and movements, such as Romantic novel or beat generation. It is even possible to discuss about the historical evolution of a certain style.
However, is it possible to define the style of a piece of text as precisely as it is possible to define its length or the color of the ink used to write it? In other words, is it possible to define some features or quantities that allow us to directly "measure" the style of a text? The answer to this question is quite elusive; though we all intuitively recognize different styles when we

see or hear them, it is difficult to say anything unified that relates the many diverse aspects of style.

Many works exist in literature, attempting to solve this problems from a computational point of view, see for example [32, 7, 4]. These works explore the nature of style, how it is perceived, and how it is used, specifically in terms of how information is represented, organized, and transformed in the production and perception of different styles. These approaches aim to point towards more general and comprehensive understandings of style and how it works, as well as to enable development of more useful intelligent computer systems.

This thesis does not want to answer directly the problem of modeling the style. Instead, assuming that there exist techniques able to capture and model the style of a text from a computational point of view, the goal of the thesis is to show how these techniques can be exploited to generate texts, and particularly poetry and lyrics, in a predefined style.

Markov processing are one of such techniques able to capture the style of a text. In Natural Language processing Markov processes (sometimes called *N-grams*, *Statistical* or *Generative Language models*) are widely used to solve many type of language processing problems, such as speech recognition, spelling error detection, document classification and retrieval sentimental analysis [20]. Several studies (see [41] for a complete survey) have shown that they are also able to capture the style of the author of a text. The basic idea is that a given author can be modeled as a Markov process. Then, given an unknown piece of text, the model can be used to compute the probability that the text is generated by the Markov process that represent the author. But this procedure can be reversed to generate new pieces of text in the style of a given author, as shown in the next chapter.

1.2 Generating texts with style: A natural language task

Since this thesis deals with the problem of generating text in natural language, it is part of a bigger research field called Natural Language Generation. This section reviews the field of natural language generation, starting with a brief history of natural language generation. Then it covers the state of the art literature about two subfields of natural language generation, that are related to the goal of the thesis: generation of text with style and poetry generation.
Natural Language Generation (NLG) is a subfield of natural language processing (NLP) concerning the generation of natural language outputs from a machine representation system such as a knowledge base or a logical form. NLG is also known as language production when such formal representations are interpreted as models for mental representations.
A NLG system can be thought as a translator that converts a computer based representation into a natural language representation. NLG may be viewed as the opposite of natural language understanding. Natural language understanding concerns the issue of disambiguating and understanding a natural language input sentence to produce the machine representation. On the contrary, natural language generation concerns the issue of putting a machine representation into words. A simple example of natural language systems are systems that generate form letters. Such systems do not typically involve complex grammar rules, but they generate letter by filling slots in a template. More complex NLG systems dynamically create texts to meet a communicative goal. As in other areas of natural language processing, this can be done using either explicit models of language (e.g., grammars) and the domain, or using statistical models derived by analyzing human-written texts.

1.2.1 A brief history of natural language generation

This section presents a brief history of natural language generation. For more details, see [36]
The first works on what is called now language generation was carried out in the 1950s and 1960s in the context of machine translation projects. The first translation systems were principally concerned with mapping sentences between languages. Also the first system that attempted to use natural language grammars as a means of randomly generating well-formed sentences wad developed in the 1960s.
In the 1970s appeared the first works on NLG which generate texts to communicate or explain non-linguistic information. These work were concerned with the problem of mapping the predetermined content of a sentence into an actual realization of that content. In 1970 also appeared the first PhD theses devoted to NLG: in particular, Goldman's research on choosing appropriate words to express abstract conceptual content [19], and Davey's [8] work on generating appropriate textual structures and referring expressions in descriptions of tic-tac-toe games. Thanks to these pioneering researches came out that language generation was not simply the reverse of language understanding: in fact important questions to be answered in natural language generation started to appear, without no obvious corollaries in the concerns of natural language understanding research.
Thanks to these understandings, the 1980s was the decade in which NLG really grew in importance: several influential PhD theses appeared in the early 1980s, most notable those of McKeown [27] and Appelt [3]. These thesis strongly impacted the field, and implicitly drove a great number of subsequent research.
the most significant trend was a move away from building large monolithic systems that attempted to perform all aspects of the generation task, towards specific NLG problems, such as syntactic realization, lexical choice, and text planning.
The initial work on the influential Penman text generation system, appeared

early in the decade [[25]]. The initial work on using Rhetorical Structure Theory in text generation appeared mid-decade [26].

The NLG research community also started to acquire an identity by holding dedicated NLG workshops. The first International Workshop on Natural Language Generation was held in 1983. By the end of the 1980s a significant number of researchers had moved into the area, and two traditions developed. Following the early work in the field, a considerable amount of research came from an *artificial intelligence* perspective. At the same time, an other considerable amount of research came more from linguistics and computational linguistics.

In the 1990s NLG has continued to grow, with many more papers, PhD theses, workshops, and available research software. Work in the field is now very diverse, and it can be difficult to grasp the nature of the field as a whole. In the 1990s also appeared the first real-world application systems that use NLG technology, including the pioneering FoG system [18], which was the first NLG system to enter everyday use. The Fog system was used by Environment Canada to generate weather forecasts in French and English in the early 1990s

In parallel with the emergence of applications has been an increased interest in the research community in applied issues such as comparing the engineering costs and benefits of different NLG techniques, and a concern with the evaluation of systems.

in the 2000s some new key themes emerged. One of them is the issue combining text generation with graphics generation. this is a realism borne of the observation that most real world documents contain diagrams and pictures as well as text, and analyzing how text compares to graphics as a communication medium raises many fundamental questions in cognitive science. There is also more interest in multilingual generation, and in finding architectures and techniques that can easily produce equivalent texts in several languages. Among all these new trend, two of them will be covered more in the details in the next two sections, as they are strongly related to the work of this thesis:

generating text with style and poetry generation.

1.2.2 Other works about text generation and style

Only a few Natural Language Generation systems have been proposed to generate *technical texts* in a given style.
[34] describes an approach for generating a wide variety of texts expressing the same content. By treating stylistic features as constraints on the output of a text planner, it explores the interaction between various stylistic features (from punctuation and layout to pronominal reference and discourse structure) and their impact on the form of the resulting text. In this work the Iconoclast system is proposed. This system generates patient information leaflets. The style is parameterized by low-level parameters, such as paragraph length or the use of specific technical terms.
Skillsum [37] generates feedback reports about people's literacy and numeracy skills. In Skillsum, the style is governed by rules based on a manual analysis of the corpus to imitate. In this work three mechanisms for incorporating style into NLG choice-making are explored: (1) explicit stylistic parameters, (2) imitating a genre style, and (3) imitating an individual's style.

1.2.3 Poetry generation

On the other hand, poetry generation seems to be a more active research field and several systems to generate poetic texts were proposed.
McGonagall [35] uses genetic algorithms to generate texts that are syntactically correct, following imposed meter patterns and which broadly convey a given meaning. The paper reports some experiments to properly handle the multi-objective optimization nature of poetry generation as a stochastic search that seeks to produce a text that simultaneously satisfies the properties of grammaticality (be syntactically correct), meaningfulness (convey a given meaning), and poeticness (follow imposed meter patterns). However,

this system is still unable to reach the goal of generating an entire poem.
The WASP system [15] and its evolution ASPERA [16] produce Spanish poetry from several input data, such as the choice of vocabulary and a poem type. WASP is a forward reasoning rule-based system that takes as input data a set of words and a set of verse patterns and returns a set of verses. Using a generate and test method, guided by a set of construction heuristics obtained from formal literature on Spanish poetry, the system can operate in two modes: either generating an unrestricted set of verses, or generating a poem according to one of three predefined structures (romance, cuarteto, or terceto). ASPERA is a system that obtains from the user the intended message, then applies a knowledge-based preprocessor to select the most appropriate metric structure for the user's wishes. By intelligent adaptation of selected examples from a corpus of verses, it carries out a prose-to-poetry translation of the given message. In the composition process, ASPERA combines natural language generation and Case Base Reasoning techniques to apply a set of construction heuristics obtained from formal literature on Spanish poetry.
In the field of lyrics generation, Tra-la-lyrics [31] automatically generates percussive lyrics to accompany a given piece of music. Tra-la-Lyrics aims to generate lyrics for given melodies, using three different strategies: random words, generative grammars or generate-and-test.
[39] describes two natural language processing systems designed to assist songwriters in obtaining and developing ideas for their craft. Titular Titular automatically generates novel song titles to inspire songwriter in lyrics production. LyriCloud is a word-level language browser which allows users to interactively select words and receive lyrical suggestions in return.
It is important to note that none of these systems address explicitly the problem of style. To our knowledge, the only systems that generates literary texts with style are the poetry generator proposed in [17] and RKCP (Ray Kurzweil's Cybernetic Poet) [22].
In [17], the style is modeled as a "blending principles choice". [17] proposes

a new approach to style, arising fromour work on computational media using structural blending, which enriches the conceptual blending of cognitive linguistics with structure building operations in order to encompass syntax and narrative as well as metaphor. The central idea is to analyze style in terms of blending principles, based on the finding that different principles from those of common sense blending are needed for some creative metaphors Although the authors claim that an existing style can be approximated by carefully tuning these blending principles, they do not explain how to do it in practice.

RKCP is the closest system that looks at solving the same problem that the one studied here, and uses Markov processes trained on a corpus of given authors to generate poems in the same style. In fact RKCP reads a selection of poems by a particular author or authors and then creates a ”language model” of that author's work. RKCP can then write original poems from that model. The generation is controlled by parameters such as the type of stanza and rhymes. However, RKCP uses a *generate-and-test* approach to validate the verses generated by the Markov processes: when writing original poems, RKCP first determines a set of ”goals” for the word, then searches for the words that will fulfill this set of goals. This selection is, however, only tentative because later consideration may cause the word to be ultimately rejected. This approach is therefore both incomplete and unbounded in terms of execution time.

However it is possible to control Markov process to directly generate verses that satisfy all the constraint imposed by the user, without the need of generate-and-test. This can be achieved by the means of Constrained Markov Processes. The next chapter will introduce this technique.

Chapter 2

Constrained Markov processes

This chapter describe constrained Markov processes, a powerful tool to generate texts in a given style that satisfy control constraints. As the name suggest, constrained Markov processes combine traditional Markov processes and constrain satisfaction techniques. The results of this combination is a new technique that borrows the best features of each part, trying to avoid the principal drawbacks of traditional Markov processes and constraint satisfaction in the framework of sequence generation. In fact Markov processes are able to generate texts that follow a given style, but without any control on the structure of such sequences. On the other hand, using constraint satisfaction it is possible to fully control the structure of the sequences to generate, but at the drawback of loosing the notion of style. Constrained Markov processes, as will be shown at the end of this chapter, ensure that the generated sequences satisfy control constraints and follows the target style.

This chapter is divide in three section. The first and the second section introduce respectively Markov processes and constraint satisfaction. The third section describes constrained Markov processes.

2.1 Markov processes: style without control

Markov processes are a popular modeling tool used in content generation applications, such as text generation, music composition and interaction. Moreover, as seen in the previous chapter, they can be used to model the style of a given given author.

Markov processes are based on the “Markov property” which states that the probability of the future state of a sequence depends only on the last state.

More formally, a Markov process is a sequence of random variables X_1, X_2, X_3, …with the Markov property. The Markov property states that, given the present state, the future and past states are independent. i.e.,

$$p(X_n = s_n|X_1 = s_1, \dots, X_{n-1} = s_{n-1}) = p(X_n = s_n|X_{n-1} = s_{n-1}) \tag{2.1}$$

The *state space* of the Markov process is the set S of all the possible values that the random variables can assume. By abuse of notation, sometimes $p(s_n|X_{n-1} = s_{n-1})$. instead of $p(X_n = s_n|X_{n-1} = s_{n-1})$. In general S is not necessarily a finite or even countable, but this thesis (and NLP in general) considers only Markov processes with finite state space.

A Markov process is said to be *time-homogeneous* if the transition probabilities are time independent, i.e.

$$p(X_n = s_n|X_{n-1} = s_{n-1}) = p(X_{n+1} = s_{n+1}|X_n = s_n). \tag{2.2}$$

Usually in NLP, if nothing else is specified, a Markov process is always supposed to be time-homogeneous. Thanks to the these properties, a Markov process can be represented by the transition matrix T, whose element are the transition probabilities $T_{ij} = p(X_n = s_j|X_{n-1} = s_i)$. A Markov can be also be described by a directed graph, whose vertex are the element of the state space and the edges are labeled by the probabilities of going from one state to the other states.

A simple example of graph representation is shown in figure 2.1. Each node represent an element in the state space $S = \{s_1, s_2, s_3\}$. According to

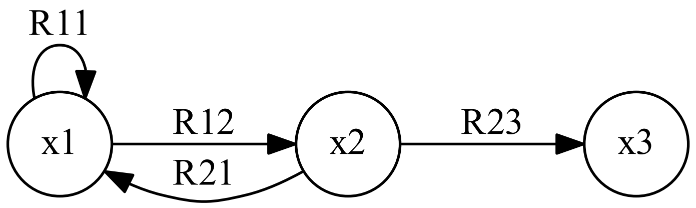


Figure 2.1: The directed graph representing a Markov process whose state space is $S = \{s_1, s_2, s_3\}$.

the graph, a s_1 is followed by itself 75% of the time, by s_2 20% of the time, and by s_3 the other 5%. The transition matrix for this Markov process is:

$$T = \begin{pmatrix} 0.75 & 0.2 & 0.05 \\ 0.1 & 0.8 & 0.1 \\ 0.5 & 0.25 & 0.25 \end{pmatrix}$$

2.1.1 Estimating a Markov process

Given a corpus of a training sequences, such as the corpus of all the texts written by a given author, there are many approaches to estimated a Markov process that model this corpus. In this thesis the *maximum-likelihood estimation* is used. Maximum-likelihood estimation estimates a Markov process from a corpus by computing the relative frequencies of each *bigram*, i.e., continuous sequence of elements that appear in the corpus. Namely, for each couple of elements s_i and s_j in the state space S the transition probability $p(s_j|s_i)$ is:

$$p(s_j|s_i) = \frac{freq(s_j, s_i)}{freq(s_i)} \tag{2.3}$$

where $freq(s_j, s_i)$ and $freq(s_i)$ count respectively how many time the bigram $s_j s_i$ and the element s_i occur in the corpus. The prior probability $p(s_i)$ of the element s_i is

$$p(s_i) = \frac{freq(s_i)}{N} \tag{2.4}$$

where N is the size of the corpus.
As an example, let us consider the following corpus simple corpus C_{ex}:

- *Clay loves Mary*
- *Mary loves Clay*
- *Clay loves Mary today*
- *Mary loves Paul today*

Applying maximum-likelihood estimation, a Markov process M_{ex} is estimated from the corpus C_{ex}. The state space of M_{ex} is $S_{ex} = \{Mary, Clay, Paul, loves, today\}$. Labeling the state space $\{1 = Mary, 2 = Clay, 3 = Paul, 4 = loves, 5 = today\}$ the transition matrix of M_{ex} is:

$$T_{ex} = \begin{pmatrix} 0 & 0 & 0 & 2/3 & 1/3 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1/2 & 1/4 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The prior vector of M_{ex} is:

$$\begin{pmatrix} 1/7 & 3/14 & 1/14 & 2/7 & 1/7 \end{pmatrix}$$

The corresponding directed graph is represented in Figure 2.2.

More sophisticated techniques, such as smoothing techniques [20], can be used, to deal with the problems caused by the sparsity of the RF estimate. In this thesis maximum-likelihood estimation is used because, in the context of text generation, it gives an acceptable estimate. However all the results discussed below are independent of the way a Markov process is estimated, and are therefore compatible with any estimation techniques.

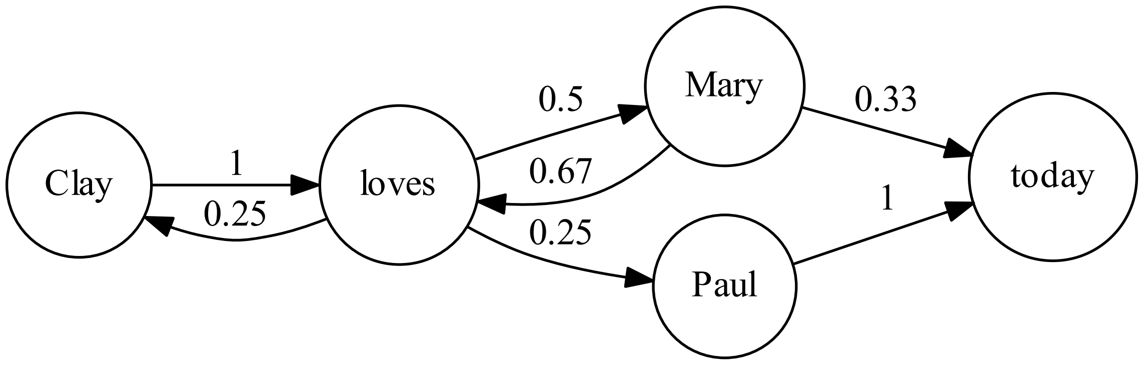


Figure 2.2: A Markov process learned from a simple corpus composed by 4 sequences. The state space of the Markov process is $S = \{Mary, Clay, Paul, loves, today\}$.

2.1.2 Generating sequence using Markov process

Once the model is learn, sequences can be generated simply by *random walk*: the first item is chosen randomly using the prior probabilities; then, a continuation is drawn using the model, and appended to the first item. This is iterated to produce a sequence of length L. This process has the advantage of being simple to implement and efficient.
Thanks to this simplicity and efficiency, Markov models are a common tool for generating content in many domains. In addition to text generation, they are widely used in music composition.
For instance, the Continuator [32] uses a Markov model to react interactively to music input. Its success was largely due to its capacity to faithfully imitate arbitrary musical styles, at least for relatively short time frames. Indeed, the Markov hypothesis basically holds for most melodies played by users (from children to professionals) in many styles of tonal music (classical, jazz, pop, etc.). The other reason of its success is the variety of outputs produced for a

given input. All continuations produced are stylistically convincing, thereby giving the sense that the system creates infinite, but plausible, possibilities from the user's style.

On the other hand, in the field of text generation, the sequence generated from a random walk are less satisfactory. Let us consider again the corpus C_{ex} used as example in the previous section. A random walk on the corresponding graph (showed in figure 2.2) could produce incorrect sequences such as "loves Mary loves Clay loves", or "Paul today". This is caused by the fact that Markov processes do not provide any control on the structure of the generated sequences. For instance, a constraint that imposes the *last word* of a 4-word sequence to be "today" and a constraint that imposes to the *first word* to rhyme with "today" create a long distance dependency between the first and the last word of the sequence. Indeed the only four-word sequences that satisfy these constraints and can be generated by M_{ex} are "Clay loves Mary today" and "Clay loves Paul today". Therefore, the first word of the sequence *must be* "Clay", excluding "Mary", "Paul", "loves" and "today" as possible first states. This implicit dependency cannot be represented in the initial Markov model.

Most approaches proposed to deal with the control issue consist in grafting heuristic search on top of random walk: simulated annealing [9], case-based reasoning [13], generate-and-test [12]. However, these methods do not offer any guarantee that a solution will be found. Moreover, the solutions found are not consistent with the underlying Markov model, i.e., the probability to draw them is not the Markov probability. Lastly, these methods are not efficient enough for interactive, real-time applications. It is important to note that Hidden Markov Models (HMM) cannot be applied, because in this context the Bellman principle does not hold. Control constraints cannot be modeled as *cumulative cost functions*: even anchor constraints (imposing a fixed value at a specific position) may have implications on the whole sequence, as shown below.

As outlined by [33], control constraints raise a fundamental issue since they

establish relationships between items that violate the Markov hypothesis: obviously as soon as the constraint *scope* (the variables on which the constraint holds) is larger than the Markov scope, the information cannot be represented in a Markov model. However, even constraints that remain within the Markov scope (e.g., unary constraints) create implicit dependencies that violate the Markov hypothesis.
[33] show that the reformulation of the problem as a constraint satisfaction problem allows, for arbitrary sets of control constraints, to compute optimal, singular solutions, i.e., sequences that satisfy control constraints while being optimally probable. However, what is often needed in practice is a *distribution* of good sequences. Therefore the approach of [33] cannot be used, as it does not produce a distribution of sequences, but only optimal solutions. Furthermore, it involves a complete search-optimization algorithm, which limits real-time use.
The last section of this chapter shows that control constraints can be "compiled" into a new stochastic process that is statistically equivalent to the initial one. This yields the advantage of retaining the simplicity of random walk, while ensuring that control constraints are satisfied. This result is obtained by establishing yet another bridge between Markov generation and constraint satisfaction. However, before tis bridge can be established, constraint satisfaction needs to be introduced.

2.2 Constraint satisfaction: control without style

An other famous technique used to generate sequences is *Constraint satisfaction*. Constraint satisfaction is the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy. In opposition to Markov processes, constraint satisfaction allows full control on the sequence to generate. However this technique does not guarantee that the generated sequences follow the desired statistical distribution.

2.2.1 Introduction to CSP

In this subsection the theory of CSP is introduced , following the presentations of [29] and [10]. A constraint satisfaction problem is defined as a set of variables that must satisfy a number of constraints. In general these constraints can affect an arbitrary number of variables, but in this work only *binary constraints* are considered.

Given two variable X_i and X_j having domains D_i and D_j, a *binary constraint* R_{ij} between these two variables is a subset of the Cartesian product of the domains:

$$R_{ij} \subseteq D_i \times D_j \tag{2.5}$$

In the case of $X_i = X_j$, $R_i i$ is a *unary constraint* on the variable X_i. A binary constraint R_{ij} can be represented using the characteristic function $\chi : D_i \times D_j \to \{0, 1\}$:

$$\chi\left(x_{i,r}, x_{j,s}\right) = \begin{cases} 1 & \text{if } \left(x_{i,r}, x_{j,s}\right) \in R_{ij} \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

Namely, assigning the values of the variables, the output of χ is 1 if these values satisfy the constraint, 0 otherwise. The outputs of χ can be arranged in a $N_i \times N_j$ matrix, where N_i and N_j are respectively the size of the domains D_i and D_j. The rows of the matrix correspond to the values of domain D_i and the columns to the values of D_j. The values of the matrix are:

$$R_{ij_{rs}} = \begin{cases} 1 & \text{if } \left(x_{i,r}, x_{j,s}\right) \in R_{ij} \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

This matrix notation will prove to be very useful in combining Markov processes and CSPs. For example, given two variables X_1 and X_2, both with domain $D_1 = D_2 = \{a, b, c\}$, let be $R_{12} = \{(a, b), (b, a), (b, c), (c, b)\}$ a binary

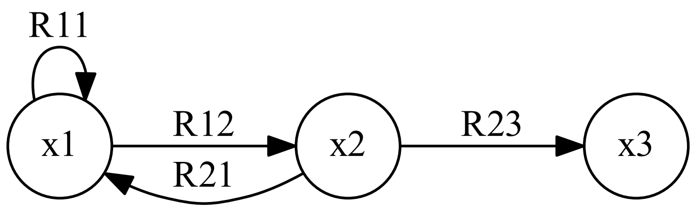


Figure 2.3: A constraint graph.

constraint among them. Therefore the matrix representation of R_{12} is

$$\begin{array}{c c} & \begin{array}{ccc} a & b & c \end{array} \\ \begin{array}{c} a \\ b \\ c \end{array} & \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

A *network of binary constraint* is a couple (X, R), where X is a set of variable $X_1, \ldots, X_n$ having domains $D_1, \ldots, D_n$, and R is a set of unary and binary constraints imposed on the variables in X. The set of solution ρ is the set of the n-tuples satisfying all the constraints, i.e.

$$\rho = \{(x_1, \ldots, x_n) \,|x_i \in D_i, (x_i, x_j) \in R_{ij}, \forall i, j\} \tag{2.8}$$

A network of binary constraint can be represented as an undirected graph, called *constraint graph.* The nodes of this graph represent the variables and the edges represent the constraints among them, i.e. two nodes are connected if a constraint imposed on their variables exists. Figure 2.3 shows an example of constraint graph.

Several operations on constraints can be defined:

Definition 2.1 (Union)**.** The *union* of two constraint R_{12} and S_{12}imposed on the same couple of variables is a new constraint U_{12} that allows all pairs allowed by either one of them.

Definition 2.2 (Intersection)**.** The *intersection* of two constraint R_{12} and S_{12} imposed on the same couple of variables is a new constraint I_{12} that allows only pairs allowed by both.

Definition 2.3 (Composition)**.** The *composition* of two constraint R_{12} and R_{23} is a new constraint R_{13} that allows a pair of values (x_1, x_3) if and only if there exists at least one value $x_2 \in D_2$ such that $(x_1, x_2) \in R_{12}$ and $(x_2, x_3) \in R_{23}$.

In matrix notation these three operation can be obtained as follows. Let be r_{ij} et s_{ij} the elements of R_{12} and S_{12}. The elements u_ij of the union $U_{12} = R_{12} \cup S_{12}$ are $u_ij = \max(r_{ij}, s_{ij})$. The elements i_ij of the intersection $I_{12} = R_{12} \cap S_{12}$ are $i_ij = \min(r_{ij}, s_{ij})$. The composition of R_{12} and R_{23} can be obtained by matrix multiplication: $R_{13} = R_{12} \cdot R_{23}$.

2.2.2 Backtrack free problems

A common algorithm for solving CSPs is the *Backtrack* search algorithm. In its simplest version, this algorithm traverses the constraint graph in a predefined order. At the i-th step it assigns values to a subsequence $(X_1, \ldots, X_i)$ of variables and attempts to append to it a new instantiation X_{i+1} such that the whole set is *consistent*. An assignment of values to a subset of variables is *consistent* if it satisfies all the constraints applicable to this subset. If no consistent assignment can be found for the next variable X_{i+1}, a dead-end situation occurs and the algorithm "backtracks" to the most recent variables, changes its assignment and continues from there. The algorithm for finding one solution is given below. It is defined by two recursive procedures, `Forward` and `Go-back`. The first tries to extend the current partial assignment, and, it it is not possible, calls the second that handles dead-end situations. Backtrack is initiated by calling `Forward` with $i = 0$, i.e. the instantiated list is empty. The procedure `Compute-candidates` selects all the values in the domain of X_{i+1} which are consistent with the previous assignments with respect to all applicable constraints $R_{j,i+1}$, $j \leq i$.

Algorithm 1 `Forward`$(x_1, \ldots, x_i)$

```
1: if $i = n$ then
2:     return the current assignment
3: end if
4: $C_{i+1} \leftarrow$ Compute-candidates $(x_1, \ldots, x_i, X_{i+1})$
5: if $C_{i+1}$ is not empty then
6:     $x_{i+1} \leftarrow$ first element in $C_{i+1}$
7:     remove $x_{i+1}$ from $C_{i+1}$
8:     Forward$(x_1, \ldots, x_i, x_{i+1})$
9: else
10:    Go-back$(x_1, \ldots, x_i)$
11: end if
```

Algorithm 2 `Go-back`$(x_1, \ldots, x_i)$

```
1: if $i = 0$ then
2:     return . No solution exists
3: end if
4: if $C_i$ is not empty then
5:     $x_i \leftarrow$ first element in $C_i$
6:     remove $x_i$ from $C_i$
7:     Forward$(x_1, \ldots, x_i)$
8: else
9:     Go-back$(x_1, \ldots, x_{i-1})$
10: end if
```

This algorithm naturally introduces a new class of problems: *Backtrack free problems.* A Backtrack-free problems is a problem for which the backtrack algorithm terminates without backtracking. More formally:

Definition 2.4 (Backtrack free problem)**.** A *Backtrack free problem* is a problem for which it exists an ordering of the variables $(X_1, \dots, X_n)$ such that, given any consistent assignment $(x_1, \dots, x_i)$ it always exists a value x_{i+1} in the domain of X_{i+1} such that $(x_1, \dots, x_i, x_{i+1})$ is consistent.

Backtrack free problems are very useful, because they can produce a solution in linear time in the number of variables, simply applying the `Forward` procedure. The remaining part of this section will show how to determine if a CSP is equivalent to a backtrack free problem and how to construct it, if exists. Two networks of constraint with the same set of variables are equivalent if they are satisfied by the same set of solutions.

The feasibility of achieving backtrack-free search relies heavily on the topology of the constraint graph. Freuder [14] has identified sufficient condition for a CSP to yield backtrack-free solution hand has shown, for example, that tree-like constraint graph can be made to satisfy these conditions with a small amount of preprocessing. The remaining part of this section studies classes of constraint graph leading themselves to back-track free solution and show efficient algorithm for solving them. Then constraints can be selectively deleted from the original specification so as to transform the original problem into a backtrack-free one.

Definition 2.5 (Ordered constraint graph [14])**.** An *ordered constraint graph* is a constraint graph in which the nodes are linearly ordered to reflect the sequence of variable assignments executed by the Backtrack search algorithm. The *width of a node* is the number of arcs that lead from that node to previous nodes, the *width of an ordering* is the maximum width of all nodes, and the *width of a graph* is the minimum width of all orderings of the graph

Figure 2.4 presents six possible orderings of a constraint graph. The width of node C in the first ordering (from the left) is 2, while the second

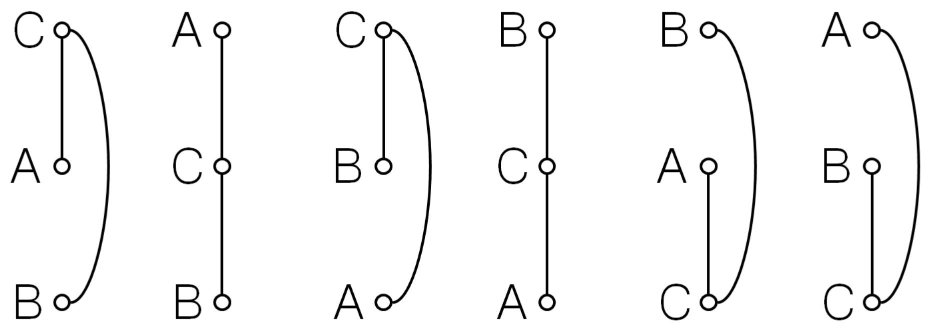


Figure 2.4: Six possible orderings of a constraint graph. The direction of instantiation goes from the bottom to the top.

ordering is 1. The width of the first ordering is 2, while the ordering of the second is 1. Therefore the width of the graph is 1. Freuder provided an efficient algorithm for finding both the width of a graph and the ordering corresponding to this width. Moreover he showed that a constraint graph is a tree if and only if its width is 1.

Montanari [29] and Machworth [23] have introduced two kinds if local consistencies among constraints named *arc consistency* and *path consistency.* Their definitions assume that the graph is directed, i.e. each symmetric constrain is represented by two directed arcs.

Let $R_{i,j}(x, y)$ stand for the proposition that (x, y) is permitted by the constraint $R_{i,j}$.

Definition 2.6 (Arc-consistency [23])**.** A directed arc (X_i, X_j) is *arc-consistent* $\Leftrightarrow$ for any value $x \in D_i$ there is a value $y \in D_j$ such that $R_{i,j}(x, y)$.

Definition 2.7 (Path-consistency [29])**.** A path of length m through nodes $(i_0, i_1, \ldots, i_m)$ is *path-consistent* if for any value $x \in D_{i_0}$ and $y \in D_{i_m}$ such that $R_{i_0,i_m}(x, y)$, there is a sequence of values $z_1 \in D_{i_1}, \ldots, z_{m-1} \in D_{i_{m-1}}$ such that

$$R_{i_0,i_1}(x, z_1) \text{ and } R_{i_1,i_2}(z_1, z_2) \text{ and } \ldots R_{i_{m-1},i_m}(z_{m-1}, y)$$

. $R_{i_0,i_m}(x, y)$ may also be the universal relation, e.g. permitting all possible pairs.

A constraint graph is arc-consistent if each of its directed arcs is arc-consistent. Similarly, a constraint graph is path-consistent is each of its paths is path-consistent. "Achieving arc consistency" means deleting certain values form the domains of certain variables such that the resulting graph is arc-consistent while representing the same overall set of solutions. To achieve path consistency, certain pairs of values that were initially allowed by the input constraints should be disallowed. Montanari and Mackworth have proposed polynomial-time algorithm for achieving arc consistency and path consistency. In [24] it is shown that arc consistency can be achieved in $O\left(ek^3\right)$ while path consistency can be achieved in $O\left(n^3k^5\right)$, where n is the number of variables, k is the number of possible values, and e is the number of edges.

Theorem 2.2.1. *Freuder [14]*

1. *If the constraint graph has a width 1 (i.e. it is a tree) and if it is arc-consistent, then it admits backtrack-free solutions.*
2. *If the width of the constraint graph is 2 and it is also path consistent, then it admits backtrack-free solutions.*

The above theorem suggests that tree-like CSPs (CSPs whose constraints graphs are trees) can be solved by first achieving arc consistency and then instantiating the variable in any width-1 order. Since this backtrack-free instantiation takes $O\left(ek\right)$ steps, and in trees $e = n-1$, the entire problem can be solved in $O\left(nk^3\right)$ [24]. The test for simplicity is also verified: it amounts to testing whether a given graph is a tree, and can be accomplished by an $O\left(n^2\right)$ spanning tree algorithm. Thus, tree-like CSPs are easy since they can be made backtrack-free after a preprocessing phase of low complexity.

The second part of the theorem tempts to conclude that a width-2 constraint graph should admit a backtrack-free solution after passing through a path consistency algorithm. In this case, however, the path consistency algorithm may add arcs to the graph and increase its width beyond 2. This happens when the algorithm disallows value pairs from nonadjacent variables (i.e. variables that were initially related by the universal constraint) and it is often the case that the passage through a path consistency algorithm makes the constraint graph complete. It may happen, therefore, that no advantage could be taken of the fact that a CSP possess a width-2 constraint graph if it is not already path consistent.

Dechter and Pearl [10] give weaker definitions of arc and path consistency, which are also sufficient to guarantee backtrack-free solutions and have two advantages over those defined by Montanari and Mackworth:

1. They can be achieved more efficiently.
2. They add fewer arcs to the constraint graph, thus preserving the graph width in a larger class of problems.

These definitions and the algorithms that exploit them are presented in the next two subsections

The case of width 1 (trees)

Achieving full arc consistency is more than is actually required for obtaining backtrack-free solutions. For example, if the constraint graph in Figure 2.5 is ordered by (X_1, X_2, X_3, X_4), nothing is gained by making the directed arc (X_3, X_1) consistent. To ensure backtrack-free assignment, it is needed only that any value assigned to the variable X_1 will have at least one consistent value in D_3. This can be achieved by making only the directed arc (X_1, X_3) consistent regardless whether (X_3, X_1) is consistent or not. Therefore arc consistency is required only with respect to a single direction, that one in which Backtrack selects variables for instantiations. This motivates the following definition.

Definition 2.8 (d-arc-consistency)**.** Given an order d on the constraint graph R, R is *d-arc-consistent* if all edges directed along d are arc-consistent.

Theorem 2.2.2. *Dechter and Pearl [10]*
Let d be a width-1 order of a constraint tree T. If T is d-arc-consistent, then the backtrack search along d is backtrack-free.

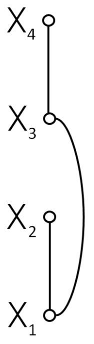


Figure 2.5: Directed constraint graph demonstrating the sufficiency of making arc (X_1, X_3) directionally consistent. The direction of instantiation goes from the bottom to the top.

Proof. Suppose that $X_1, X_2, \ldots, X_k$ where already instantiated. The variable X_{k+1} is connected to at most one previous variable (from the width-1 property), say X_i, which was assigned the value x_i. Since the directed arc (X_i, X_{k+1}) is along the order d, its arc consistency implies the existence of a value x_{k+1} such that the pair (x_i, x_{k+1}) is permitted by the constraint $R_{i(k+1)}$. Thus the assignment of X_{k+1} is consistent with all previous assignments. Reasoning by induction over k proves the theorem. □

An algorithm for achieving directional arc consistency for any ordered constraint graph is given next (the order $d = (X_1, X_2, \ldots, X_n)$ is assumed).

Algorithm 3 `DAC` (d-arc-consistency)

```
for $i = n$ to 1 by $-1$ do
  for all arc $(X_j, X_i)$; $j < i$ do
    revise$(X_j, X_i)$
  end for
end for
```

The algorithm algorithm `revise`(X_j, X_i), given in [23] deletes values from the domain D_j until the directed arc (X_j, X_i) is arc-consistent. The algorithm is proved to achieves d-arc-consistency if upon termination, any arc (X_j, X_i)

Algorithm 4 `revise`(X_j, X_i)

```
1: for all $x \in D_j$ do
2:    if there is no value $y \in D_i$ such that $R_{ji}(x, y)$ then
3:       delete $x$ from $D_j$
4:    end if
5: end for
```

along d ($j < i$) is arc consistent. The algorithm revises each d-directed arc once. It remains to be shown that the consistency of an already processed arc is not violated by the processing of subsequent arcs. Let arc (X_j, X_i) ($j < i$) be an arc just processed by `revise`(X_j, X_i). To destroy the consistency of (X_j, X_i) some values should be deleted from the domain of X_i during the continuation if the algorithm. However, according to the order by which `revise` is preformed, only lower indexed variables may have their set of values updated. Therefore, once a directed arc is made arc-consistent its consistency will not be violated.

The advantage of the directed-arc-consistency algorithm is that each arc is processed exactly once. The complexity of the algorithm is optimal, because even to verify directed arc consistency each arc should be inspected once, and that takes k^2 tests. Note that when the constraint graph is a tree, the complexity of the directional-arc-consistency algorithm is $O(nk^2)$.

Theorem 2.2.3. *Dechter and Pearl [10]*
A tree-like CSP can be solved in $O(nk^2)$

Proof. If the constraint graph is a tree, finding an order that will render it of width 1 takes $O(n)$ steps. A width-1 constraint tree can be made d-arc-consistent in $O(nk^k)$ steps, using the `DAC` algorithm. Finally, the backtrack-free solution on the resultant tree is found in $O(nk)$ steps. Summing up, finding a solution to tree-like CSPs takes $O(nk) + O(nk^2) + O(n) = O(nk^2)$. This complexity is also optimal since any algorithm for solving a tree-like problem must examine each constraint at least once, and each such examination may take, in the worst case, k^2 steps, especially when when no solution exists and the constraints permit very few pairs of values. □

The case of width 2

Order information can also facilitate backtrack-free search on width-2 problems by making path consistency algorithm directional

Montanari had shown that, if a network of constraints is consistent with respect to all path of length 2 then it is path consistent. This section will show that directional path consistency with respect to length-2 path is sufficient for ensuring backtrack-free search on width-2 problems.

Definition 2.9 (d-path-consistency)**.** A constraint graph R is *d-path-consistent* with respect to ordering $d = (X_1, X_2, \dots, X_n)$ if for every pair of values (x, y), $x \in X_i$ and $y \in X_j$ such that $R_{ij}(x, y)$ and for every $k > i, j$, there exists a value $z \in X_k$ such that $R_{ik}(x, z)$ and $R_{kj}(z, y)$.

Theorem 2.2.4. *Let d be a width-2 ordering of a constraint graph R. If R is directional arc and path consistent with respect to d, then it is backtrack-free.*

Proof. To ensure that a width-2 ordered constraint graph is backtrack-free it is required that each variable selected for instantiation will have some values consistent with all previous chosen values. Suppose $X_1, X_2, \dots, X_k$ were already instantiated. The width-2 property implies that variable X_{k+1} is connected to at most two previous variables. If it connected to X_i and X_j, $i, j < k$, then directional path consistency ensures that for any assignments of values to X_i, X_j there exists a consistent assignment for X_{k+1}. If X_{k+1} is connected to one previous variable, then directional arc consistency ensures the existence of a consistent assignment. □

An algorithm for achieving directional path consistency on ordered graphs must manage not only the changes made to the constraints, but also the changes made to the graph, i.e. the addition of new arcs. To describe the algorithm constraints are represented using the matrix representation, with a diagonal matrix R_{ii} representing the set of values permitted for the variable

X_i. The algorithm is described using the operations of intersection and composition, writing $R'_{ij} \cap R''_{ij}$ for the intersection of R'_{ij} and R''_{ij}.

Given a network of constraint $R = (V, E)$ and an ordering $d = (X_1, \dots, X_n)$, the following algorithm achieves path consistency and arc consistency relative to d.

Algorithm 5 `DPC`: d-path-consistency

```
1: $Y = R$
2: for k=n to 1 by $-1$ do
3:     $\forall i < k$ connected to $k$ do $Y_{ii} \leftarrow Y_{ii} \cap Y_{ik} \cdot Y_{kk} \cdot Y_{ki}$
4:     $\forall i, j < k$ s.t. $(X_i, X_k), (X_j, X_k) \in E$ do $Y_{ii} \leftarrow Y_{ii} \cap Y_{ik} \cdot Y_{kk} \cdot Y_{ki}$,
       $E \leftarrow E \cup (X_i, X_j)$
5: end for
```

Step 3 is equivalent to *revise*(i, k), and performs directional arc consistency. Step 4 starts after completing step 3 for $i < k$. Step 4 updates constraints between pairs of variables transmitted by a third variable which ranks higher in d. If X_i, X_j, $i, j < k$ are not connected to X_k, then the constraint between the first two variables is not affected by X_k. If only one variable X_i is connected to X_k, the effect of X_k on the constraint (X_i, X_j) will be computed by the step 4 of the algorithm. The only time a variable X_k affects the constraint between a pair of earlier variables is when it is connected to both, and it is in this case that a new arc may be added to the graph. The `DPC` algorithm will connect any parent node having a common successor. note that the convenience of writing the algorithm using matrix notation does not imply that it should be implemented that way. In fact representing constraints as sets of compatible pairs of values is easier and often requires less space. The complexity of the directional path consistency algorithm is $O(n^3k^3)$. The number of times the inner loop 4 is executed for variable X_i is at most $O(\deg^2(i))$ (the number of different pairs of parents of i), and each step is of order k^3. The computation of loop 3 is completely dominated by the computation of step 4 and can be ignored. Therefore, the

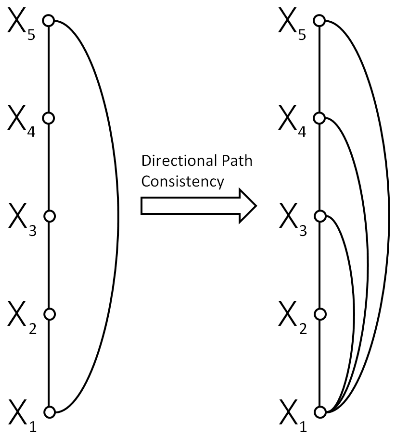


Figure 2.6: A ring is a regular width-2 CSP, therefore it can be solved in $O(n^3k^3)$.

overall complexity is

$$\sum_{i=2}^{n} \left(k^3 \cdot \deg^2(i)\right) = O(n^3k^3).$$

Applying directional path consistency to a width-2 graph may increase its width and therefore does not guarantee backtrack-free solutions. Consequently, it is useful to define the following subclass of width-2 problems.

Definition 2.10 (Regular width-2)**.** A constraint graph is *regular width-2* if there exists a width-2 ordering of the graph which remains width-2 after applying the d-path-consistency algorithm `DPC`.

A ring constitutes an example of a regular width-2 graph. Figure 2.6 shows an ordering of the ring's nodes and the graph resulting from applying the `DPC` algorithm to the ring. Both graph are of width-2.

Theorem 2.2.5. *Dechter and Pearl [10].*
A regular width-2 CSP can be solved in $O(n^3k^3)$

Proof. A regular width-2 problem can be solved by first applying the `DPC` algorithm and then performing a backtrack-free search on the resulting graph. The first takes $O(n^3k^3)$ steps and the second $O(ek)$ steps. □

The nice feature of regular width-2 CSPs is that they can be easily recognized. The relationship between the width of the graph and the tractability of the problem can be further generalized to higher widths and higher levels of consistency, see [10]. The next subsection presents one such extension.

2.2.3 A general strategy for achieving backtrack-free solutions

Both arc and path consistency cab thought of as preprocessing procedures that install uniform levels of consistency on the constraint graph. A natural generalization of these method is *K-consistency* defined by Freuder [14]. K-consistency implies the following condition. Choose ant set of $K-1$ variables along with values for each that satisfy all the constraint among them. Now choose any Kth variable. There exists a value for the Kth variable such that the K values taken together satisfy all constraints among the K variables. Freuder has shown that the a K-consistency CSP having a width-$(K-1)$ ordering admits a backtrack-free solution in that ordering. This suggest that one should be able to preprocess a CSP for a backtrack-free solution by passing it through an algorithm establishing K-consistency. However, since algorithms that achieve K-consistency change the width of the graph, it is difficult to determine in advance the level of K desired.

An alternative generalization called *adaptive consistency* [10] is specially suited for directional consistency. Adaptive consistency lets the evolving structure of directional constraint graph dictate the amount of processing required for each node.

Adaptive consistency processes order in decreasing order. Given an ordering d, adaptive consistency lets each variable impose consistency among i variables which precede it and are connected to it at the time of processing. The size of this set represent the current width of the node. The variable X_i *imposes k-consistency* on a set of $k-1$ variables if any $(k-1)$ tuple of the set is consistent with at least one value of X_i. The procedure is illustrated in Figure 2.7. Consider the constraint graph of on the left of fig 2.7. Adap-

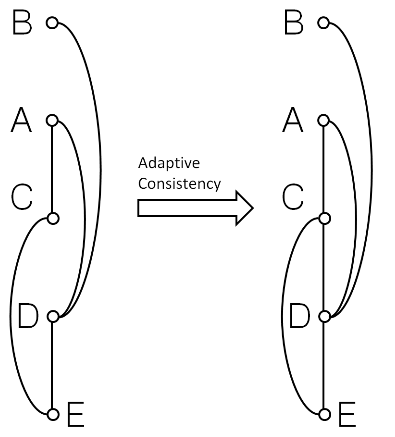


Figure 2.7: An ordered graph before (on the left) and after (on the right) being preprocessed by Adaptive consistency. Note that a constraint is added between C and D

tive consistency in the order (E, D, C, A, B) starts at node B and, having width 1, B imposes 2-consistency on D (i.e. establishes arc consistency on (D, B)). The domain of D is thus tightened. When A is processed next, it imposes 3-consistency on (C, D) (A width is 2) which may amount to adding a constraint and an arc between C and D (as in the graph on the right in figure 2.7). When the algorithm reaches node C, C's width is 2 and, therefore, a 3-consistency needs to be imposed on (E, D) but since the arc between (E, D) already exists, the effect would be merely tighten this constraint. The resulting graph is shown on the fight side of figure 2.7.

Formally, adaptive consistency can be described by the following procedure. For each variable X, `parents(`X`)` is the set of all predecessor of X currently connected to it. The parent set of each variable is computed only when it needs to be processed.

The procedure `consistency(`X`,` SET records a constraint among the variables in SET, induced by X. In other words, those instantiations of variables in SET consistent with at least one consistent value of X are retained, while the rest are discarded. To determine consistency, the procedure consider *all* constraints, old and new, which were generated up to this point

Algorithm 6 `adaptive-consistency`($X_1, \ldots, X_n$)

```
1: for r=n to 1 by −1 do
2:     compute parents($X_r$)
3:     perform consistency($X_r$, parents($X_r$))
4:     connect by arcs all elements in parents($X_r$) (if they are not yet connected)
5: end for
```

by the algorithm, and which are applicable to the variables in $V \cup SET$. A constraint is considered applicable if it involves variable X and a subset (possibly empty) of variables from SET. In figure 2.7, for instance, when `consistency(`C`,` $\{D, E\}$ is executed, the applicable constraints are the input constraint (C, E) and the recorded constraint between C and D. These two constraints will be considered to determine the set of value pairs of (D, E) consistent with at least one value of C.

In general, an ordered constraint graph will be backtrack-free if for every subset of consistently instantiated variables $(X_1, \ldots, X_r)$ there exists a value for X_{r+1} which is consistent with the current instantiation of $(X_1, \ldots, X_r)$.

Theorem 2.2.6. *Dechter and Pearl [10].*
An ordered constraint graph processed by adaptive consistency is backtrack-free.

Proof. Suppose that the first r variables were already instantiated, and assume that X_{r+1} has a parent of size k, $(X_1, \ldots, X_k)$. This parent set was identified and processed by adaptive consistency, namely the procedure `consistency(`X_{r+1}`,` $(X_1, \ldots, X_k)$`)` recorded a constraint on the k variables ensuring that any k-tuple which is not consistent with at least one consistent value of X_{r+1} is discarded. Therefore, any consistent k-tuple having passed this filter, has a consistent extension in X_{r+1}. □

An important feature of the adaptive consistency technique is that, unlike non directional K-consistency method, it permits the calculation of a bound on its complexity prior to conducting the search. Let $W(d)$ be the width associated with the ordering d and $W^*(d)$ the width of the graph induced by adaptive consistency in that ordering. The worst-case complexity of adaptive consistency along d is $2^{W^*(d)+1}$ since the `consistency` procedures records a constraint on $W^*(d)$ variables induced by their common successor, a process comparable to solving CSP with $W^*(d) + 1$ variables. $W^*(d)$ can be found in $O(n + e)$ time [42] prior to the actual processing.

2.3 Constrained Markov processes: style and control

In the previous two sections Markov processes and Constraint satisfaction are introduced. Markov processes can be used to generate sequences that follows a given style but without control on their structure. On the other hand Constraint satisfaction can be used to generate sequences that satisfy control constraints that impose a given structure, but loosing style. In this section *constrained Markov processes* are introduced as a combination of traditional Markov processes and constraint satisfaction, to fully solve the problem of generating sequences that follow a given style and satisfy control constraint at the same time.

2.3.1 Problem Statement

This section will show how to generate fixed-length sequences from a Markov process M that satisfy control constraints. To use a random walk approach, a Markov process $\tilde{M}$ that generates exactly the sequences satisfying the control constraints with the probability distribution defined by M. In general, it is not possible to find such a Markov process because control constraints violate the Markov property, as outlined by [33]. However, when control constraints remain within the Markov scope, the rest of the section will shows that such a model exists and can be created with a low complexity.

A Markov process M is defined over a finite state space $S = \{s_1, \ldots, s_n\}$. A sequence s of length L is denoted by $s = s_1, \ldots, s_L$ with $s_i \in A$. S^L is the set of all sequences of length L generated by M with a non-zero probability:

$$p_M(s) = p_M(s_1) \cdot p_M(s_2|s_1) \cdots p_M(s_L|s_{L-1}) \tag{2.9}$$

Following [33], the sequence to generate is represented as a sequence of finite-domain constrained variables $\{V_1, \ldots, V_L\}$, each with domain S, and Markov properties as *Markov constraints* on these variables, defined below. Control constraints are also represented as finite-domain constraints. The induced CSP is denoted by $\mathcal{P}$ and the set of solutions S_C. Given these notations, $\tilde{M}$ should verify:

(I) $p_{\tilde{M}}(s) = 0$ for $s \notin S_C$,

(II) $p_{\tilde{M}}(s) = p_M(s|s \in S_C)$ otherwise.

These properties state that $\tilde{M}$ generates exactly the sequences $s \in S_C$. Most importantly, sequences in S_C have the same probabilities in M and $\tilde{M}$ up to a constant factor $\alpha = p_M(s \in S_C)$, i.e., $\forall s \in S_C, p_{\tilde{M}}(s) = 1/\alpha \cdot p_M(s)$.

The main result presented in this chapter is that for a certain class of induced CSPs, hereafter referred to as *Binary-Sequential CSPs* (BSC), there exists a *non-homogeneous* Markov process $\tilde{M}$ that satisfies (I) and (II). The scope of a constraint is defined as the interval between its leftmost and rightmost variables. A Binary-Sequential CSP is a CSP that contains only constraints whose scope remains within the scope of the Markov order. With a Markov order of 1, these constraints consist in 1) unary constraints and 2) binary constraints among adjacent variables (see Section 2.3.2).

A non-homogeneous Markov process (NHM) is a Markov process whose transition matrices change over time [21]. A NHM of length L is defined as a series of transition matrices $\tilde{M}^{(i)}$, $i = 0, \ldots, L-1$.

The next subsections will describe how to build $\tilde{M}$ from M and its induced BSC, and show that $\tilde{M}$ achieves the desired properties.

2.3.2 Construction of $\tilde{M}$

$\tilde{M}$ is obtained by applying two successive transformations to the initial model M. The first transformation exploits the induced CSP to filter out state transitions that are explicitly or implicitly forbidden by the constraints. This is achieved by replacing the corresponding transition probabilities by zeros in the initial transition matrices. A side-effect is that the transition matrices are not stochastic anymore (rows do not sum up to 1 any longer). The second transformation consists in renormalizing those matrices to obtain a proper (non-homogeneous) Markov model, a step which turns out to be non trivial. Thanks to this step, the resulting model satisfies (I) and (II).

Induced CSPs

Let us consider a BSC with unary control constraints $U_1, \ldots, U_L$ and binary constraints $B_1, \ldots, B_{L-1}$. U_i defines the states that can be used at position i in the sequence. B_i defines the allowed state transitions between positions i and $i+1$. Markov constraints, denoted by $K_1, \ldots, K_{L-1}$, are posted on all pairs of adjacent variables. Markov constraints represent the following relation:

$$\forall i, \forall a, b \in A, K_i = true \Leftrightarrow p_M(b|a) > 0.$$

The first step of the process is to make $\mathcal{P}$ directional arc-consistent, using the algorithm 3, described in the previous section.

It is important to note here that enforcing directional arc-consistency on a BSC $\mathcal{P}$ is sufficient to allow the computation of the transition matrices once for all, prior to the generation, with no additional propagation. This can be shown as follows:

Proposition: If $\mathcal{P}$ is arc-consistent, then for all consistent partial sequences $s_1 \ldots s_i$ (i.e., sequences that satisfy all the constraints between variables $V_1, \ldots, V_i$), the following properties hold:

(P1) $\exists s_{i+1} \in \mathcal{D}(V_{i+1})$ such that $s1 \ldots s_i s_{i+1}$ is consistent.

(P2) $s1 \dots s_i s_{i+1}$ is consistent $\Leftrightarrow s_i s_{i+1}$ is consistent.

Proof. $\mathcal{P}$ is of width 2 as its constraint network is a tree; therefore arc-consistency enables a backtrack-free resolution and e every partial consistent sequence can be extended to a solution, which is equivalent to P1. The condition in P2 is obviously sufficient; it is also necessary as no constraint links V_{i+1} back to any other variable than V_i. $\square$

P1 and P2 ensure that the domains of variables after arc-consistency contain exactly the valid prefixes and suffixes of the corresponding transitions. Note that arc-consistency of Markov constraints as such solves the zero-frequency problem, regardless of control constraints: no choice made during the random walk can lead to a zero-frequency prefix.
The next step is to extract the matrices from the domains.

2.3.3 Extraction of the Matrices

Recall that goal is to generate a non-homogeneous Markov model, represented by a series of transition matrices. An intermediary series of L matrices $Z^{(0)}, \dots, Z^{(L-1)}$ are obtained by zeroing, in the initial matrix, the elements that correspond to values or transitions that were removed during arc consistency. More precisely, the procedure is:

- Inizialization:
  $Z^{(0)} \leftarrow M_0$ (the prior probabilities of M),
  $Z^{(i)} \leftarrow M$, $\forall i = 1, \dots, L-1$ (the transitions).

- For each $a_k \in A$ removed from the domain of V_i:
  $Z^{(i)}_{j,k} \leftarrow 0$, $\forall j = 1, \dots, n$ (set the k-th column to zero).

- All forbidden transitions in the binary constraints should also be removed from the matrices:
  $Z^{(i)}_{j,k} \leftarrow 0$, $\forall i, j, k$ such that $B_i(a_j, a_k) = false$.

Modifying the initial matrix makes it non stochastic anymore. The next section describes how to renormalize the matrices to satisfy property (II).

2.3.4 Renormalization

The final transition matrices $\tilde{M}^{(i)}$ of $\tilde{M}$ are build from the intermediary matrices $Z^{(i)}$. Transition matrices could be renormalized *individually*, i.e., by dividing each row by its sum. This would indeed produce an non-homogeneous Markov model, but this model does not satisfy property (II) above, as it generates sequences with a different probability distribution than M.

The normalization process should indeed maintain the initial probability distribution. It turns out that a simple right-to-left process can precisely achieve that. The idea is to back propagate the perturbations in the matrices induced by individual normalization, starting from the right-most one.

To do this, first the last matrix $Z^{(L-1)}$ is normalized *individually* . Then the normalization is propagated from right to left, up to the prior vector $Z^{(0)}$. The elements of the matrices $\tilde{M}^{(i)}$ and the prior vector $\tilde{M}^{(0)}$ are given by the following recurrence relations:

$$\begin{aligned} \tilde{m}_{j,k}^{(L-1)} &= \tfrac{z_{j,k}^{(L-1)}}{\alpha_j^{(L-1)}},\ \alpha_j^{(L-1)} = \sum_{k=1}^{n} z_{j,k}^{(L-1)} \\ \tilde{m}_{j,k}^{(i)} &= \tfrac{\alpha_k^{(i+1)} z_{j,k}^{(i)}}{\alpha_j^{(i)}},\ \alpha_j^{(i)} = \sum_{k=1}^{n} \alpha_k^{(i+1)} z_{j,k}^{(i)} \quad 0 < i < L-1 \\ \tilde{m}_{k}^{(0)} &= \tfrac{\alpha_k^{(1)} z_k^{(0)}}{\alpha^{(0)}},\ \alpha^{(0)} = \sum_{k=1}^{n} \alpha_k^{(1)} z_k^{(0)} \end{aligned} \tag{2.10}$$

By construction, when $\alpha_j^{(i)} = 0$, the j-th columns of the preceding $Z^{(i)}$ contain only 0 as well. By convention, the division yields 0 since there is no normalization to back propagate. These coefficients can be computed in $O(L \times n^2)$. It is now possible to show that this model satisfies the 2 desired properties.

Proposition: The $\tilde{M}^{(i)}$ are stochastic matrices and the non-homogeneous Markov process $\tilde{M}$ defined by the $\tilde{M}^{(i)}$ matrices and the prior vector $\tilde{M}^{(i)}$ satisfies (I) and (II).

Proof. The $\tilde{M}^{(i)}$ matrices are stochastic by construction, i.e., each row sums up to 1. The probability of a sequence $s = s_1 \dots s_L$ to be generated by $\tilde{M}$ is:

$$\begin{aligned} p_{\tilde{M}}(s) &= p_{\tilde{M}}(s_1) \cdot p_{\tilde{M}}(s_2|s_1) \cdot \ldots \cdot p_{\tilde{M}}(s_L|s_{L-1}) \\ &= \tilde{m}^{(0)}_{k_1} \cdot \tilde{m}^{(1)}_{k_1,k_2} \cdot \ldots \cdot \tilde{m}^{(L-1)}_{k_{L-1},k_L} \\ &= \tfrac{1}{\alpha^{(0)}} \cdot z^{(0)}_{k_1} \cdot z^{(1)}_{k_1,k_2} \cdot \ldots \cdot z^{(L-1)}_{k_{L-1},k_L}, \end{aligned} \tag{2.11}$$

where k_i is the index of s_i in A. Hence, by construction of $Z^{(i)}$:

(I) $p_{\tilde{M}} = 0$ for $s \notin S_C$,

(II) $p_{\tilde{M}} = 1/\alpha^{(0)} \cdot p_M(s)$ otherwise.

$\square$

$\alpha^{(0)}$ is precisely the probability for sequences in M of satisfying the control constraints, i.e., $\alpha^{(0)} = p_M(S_C)$.

2.3.5 A simple example

Let us consider a Markov model M estimated from the simple corpus C_{ex} already used in the first section. To recall it, the corpus is composed by the following 4 sequences:

- *Clay loves Mary*
- *Mary loves Clay*
- *Clay loves Mary today*
- *Mary loves Paul today*

The transition matrix of M is:

$$T_{ex} = \begin{pmatrix} 0 & 0 & 0 & 0.67 & 0.33 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0.5 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

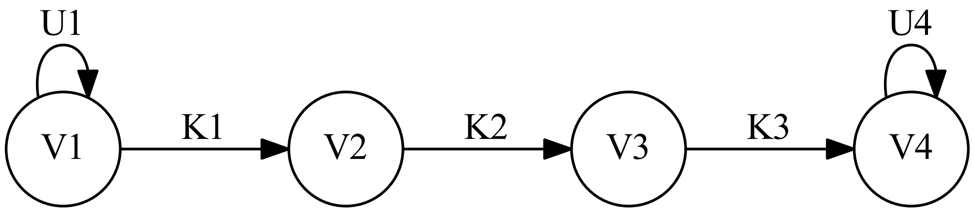


Figure 2.8: The constraint graph related to the CSP used in the example.

The prior vector of M is:

$$(1/7 \quad 3/14 \quad 1/14 \quad 2/7 \quad 1/7)$$

For simplicity, in the rest of this section, each words in the corpus is denoted only by its initial, i.e. $M = Mary$, $C = Clay$, $P = Paul$, $L = loves$, $T = today$.
This example consider the problem of generating sequences of 4 words. There are 14 possible such sequences with non-zero probabilities. For instance, sequence *Clay loves Mary loves* has probability:

$$p_M(CLML) = p_M(C)p_M(L|C)p_M(M|L)p_M(L|M) = \frac{1}{14}.$$

The following control constraint are imposed: the first word has to rhyme with the word “today” and the last word has to be exactly “today”. Translated in the constraint satisfaction formalism, the constraints are:

$$U_1 = \{Clay, today\},$$

$$U_4 = \{today\}.$$

Figure 2.8 shows the corresponding constraint graph. Once the induced CSP is defined, it is possible to apply the directional arc consistency algorithm on it to made it backtrack-free. Once the problem is backtrack free, it possible

to extract the corresponding Z matrices as explained in section 2.3.3.
The matrices obtained for the example are the following:

$$Z^{(0)} = \begin{pmatrix} 0 & 3/14 & 0 & 0 & 0 \end{pmatrix}$$

$$Z^{(1)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Z^{(2)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0.5 & 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Z^{(3)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0.33 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The final step consists in normalize these Z matrices to obtained he final matrices for the example:

$$\tilde{M}^{(0)} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$\tilde{M}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\tilde{M}^{(2)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2/5 & 0 & 3/5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\tilde{M}^{(3)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The only two sequence allowed by the constraints imposed are $Clay\ loves\ Mary\ today$ and $Clay\ loves\ Paul\ today$. It easy to see that these two sequences are the only that $\tilde{M}$ can generate. Moreover $\alpha = \frac{5}{56}$, in fact $P_M(CLMT) = 1/28 = 5/56 * 2/5 = \alpha P_{\tilde{M}}(CLMT)$ and $P_M(CLPT) = 3/56 = 5/56 * 3/5 = \alpha P_{\tilde{M}}(CLPT)$, as expected.

2.3.6 Generalization to non binary-sequential CSP

The process described in this chapter can be extended quite easily to generic induced CSP, non necessarily binary-sequential. This is done by first applying the adaptive consistency algorithm described in the previous section on the direction $d = 1, 2, \dots, L$, to made the CSP backtrack-free. Then the normalization is again back-propagated starting from the last variable V_L. The difference only is that, from a variable V_i, the normalization is back-propagated to all the variables connected to them, not only to V_{i-1}. The resulting stochastic process will not be a non-homogeneous Markov process anymore, but a Bayesian network. However, applying adaptive consistency can be very time consuming, due to its exponential computational complexity. Moreover most of the control required to generate well formed poetic texts can be expressed using a clever combination of unary constraints as will be showed in the next chapters.

Chapter 3

Constraints for Poetry generation

In the last chapter constrained Markov processes are introduced. These kind of processes can be used to generated sequences that satisfy control constraints. This chapter explains how constrained Markov processes can be used to generate poetry and lyrics, by showing how high level features of a poetic text can be represented by meaning of of unary constraint. Following the model of poetry generation introduced in [35], the following classes of text features are chosen to be represented:

1. Rhyme
2. Meter [1]
3. Syntactic correctness
4. Semantic Relatedness

These four classes of features are listed from the easiest to the harder to represent.
Meter and rhyme are the easiest features because they are objective feature. Two verses rhyme or not according to the phonetics of the last syllables,

[1]Actually in [35] rhyme and meter are grouped together in the "poeticness" class.

which are well coded and listed in pronunciation dictionaries. Similarly it is easy to count the number of syllables of a verse and the position of the stressed vowels.
Ensuring the syntactic correctness of a generated verses is harder. In fact, there is no simple algorithm to ensure that a verse or a sequence of verses are correctly well formed in the language of the text. Moreover, in poetry and lyrics generation the frontier between correct and incorrect verses is particularly fuzzy. However the combination of Markov probabilities and a simple *template-based* approach [11] suffices to enforce syntactic correctness is enforced, as will be shown later in the chapter.
Semantic relatedness is the hardest features to achieve. In fact, understanding the meaning of a text, and moreover writing a text that cover an aimed meaning is a difficult task even for humans. Therefore a simple approach is chosen, assuming that a certain verses cover a given topic if it contains words that are semantically related to the topic. This is, of course, an assumption too restrictive, but in practice it is enough to ensure that generated verses are semantically related to the aimed topics. The next subsections describe on details the constraints that enforce all these features and can be used to generate poetic text.

3.1 Rhyme Constraints

A rhyme is a repetition of similar sounds in two or more words, most often at the end of lines in poems and songs. Since the pronunciation of a word in a given language, such as English or Italian, is a well coded and objective feature, it is natural to represent rhymes unary constraints on the ending words of verses. Given a target word s, a *rhyme constraint* is satisfied by all the words in the corpus that rhyme with s, according to its phonetic spelling.
Since this thesis focus on the generation of texts in English the Carnegie Mellon University (CMU) pronunciation dictionary [38] is chosen to extract the

phonetic spelling of words. The CMU pronouncing dictionary is a machine-readable pronunciation dictionary for North American English that contains over 125,000 words and their transcriptions. This format is particularly useful as it has mappings from words to their pronunciations in the given phoneme set. The current phoneme set contains 39 phonemes, for which the vowels may carry lexical stress:

0 Non stressed vowel

1 Stressed stress [2]

The complete list of 39 phonemes is given in table 3.1. Two words are said to rhyme if the suffixes of their spellings from the last stressed vowels are the same. For example "today" ([T, AH0, D, EY1]) rhymes with "pray" ([P, R, EY1]). More formally, lets PH the set of all phonemes. Given the word w, $pr(w) = ph_1 \dots ph_n$, $ph_i \in C$ is the pronunciation of w according to the CMU pronunciation dictionary. The *rhyme suffix* of w is $rhy(w) = ph_k \dots ph_n$, where k is the position of the last stressed vowels, i.e. the position of the last symbol tagged with 1 (note that in the CMU dictionary the consonant are not tagged). Therefore two words w and v rhymes if $rhy(w) = rhy(v)$. This leads to the following definition of *rhyme constraint.*

Definition 3.1 (Rhyme constraint)**.** Given a word w, the *rhyme constraint* R_w^{rhy} imposed by w, is the unary constraint defined by the set

$$R_w^{rhy} = \{v | rhy(w) = rhy(v)\}.$$

3.2 Rhythmic Templates

In poetry, meter is the basic rhythmic structure of a verse, and it is defined by the number and position of the stressed syllables in a verse. Using

[2]Actually the CMU pronunciation dictionary contains also vowels tagged with the symbol 2 (Secondary stress), but for simplicity in this thesis all the stressed vowels are tagged with symbol 1.

Table 3.1: The complete list of the 39 phonemes used by the CMU pronunciation dictionary.

Phoneme	Example	Translation	Phoneme	Example	Translation
AA	odd	AA D	AE	at	AE T
AH	hut	HH AH T	AO	ought	AO T
AW	cow	K AW	AY	hide	HH AY D
B	be	B IY	CH	cheese	CH IY Z
D	dee	D IY	DH	thee	DH IY
EH	Ed	EH D	ER	hurt	HH ER T
EY	ate	EY T	F	fee	F IY
G	green	G R IY N	HH	he	HH IY
IH	it	IH T	IY	eat	IY T
JH	gee	JH IY	K	key	K IY
L	lee	L IY	M	me	M IY
N	knee	N IY	NG	ping	P IH NG
OW	oat	OW T	OY	toy	T OY
P	pee	P IY	R	read	R IY D
S	sea	S IY	SH	she	SH IY
T	tea	T IY	TH	theta	TH EY T AH
UH	hood	HH UH D	UW	two	T UW
V	vee	V IY	W	we	W IY
Y	yield	Y IY L D	Z	zee	Z IY
ZH	seizure	S IY ZH ER			

the the CMU dictionary it is easy to extract and represent the meter of a verse, by labeling each word with a *rhythmic tag*. The rhythmic tag $rtm(w)$ of a word w is a sequence of 0 and 1, defined by the sequence of the lexical stresses of the vowels composing the word. For instance, the rhythmic tag of "today" [T, AH0, D, EY1] is $rtm(\text{today}) = 01$. The meter of a verse is then the list of the rhythmic tags of its words. For instance the meter of the verse "Innocence of a story I could leave today" is [101, 1, 1, 10, 1, 1, 1, 01].
Meter can be imposed on a verse by a sequence of unary constraints, called a *rhythmic template*. A rhythmic template is a sequence of unary constraints, called *rhythmic constraint* that impose the rhythm on the underlying constrained words.

Definition 3.2 (Rhythmic constraint)**.** Given a rhythm tag t, the *rhythmic constraint* R_t^{rtm} imposed by t, is the unary constraint defined by the set

$$R_t^{rtm} = \{w | rtm(w) = t\}\,.$$

A *rhythmic template* is a sequence of rhythmic constraints.

3.3 Syntax: Part-of-Speech Templates

A poetic text is of course not simply a concatenation of words that satisfy some formal requirements, but must be syntactically well- formed and convey some meaningful message. This section shows as unary constraints are well-adapted to ensure syntactical correctness.

Syntactic correctness is enforced by the combination of Markov probabilities and a *template-based* approach [11]. A *part-of-speech template* is a sequence of *part-of-speech* (POS).

A part of speech (also known as a word class, a lexical class, or a lexical category) is a linguistic category of words (or more precisely lexical items), which is generally defined by the syntactic or morphological behavior of the lexical item in question. Common linguistic categories include noun and verb, among others.

Each word in the state space is tagged with one or more part-of-speech . For instance the word “run” is tagged with the POS Noun and Verb. A part of speech template is then a sequence of part-of-speech tags. The idea behind part-of-speech templates is that a well constructed template can be used to generate verses that will be syntactically well formed. For example the template [PRP, VBD, IN, PRP, RB] (Personal pronoun, Verb past tense, Preposition, Adverb) is satisfied by the verse “she knocked upon it anyway”.

More formally, let POS be the set of all available part-of.speech tags. Given a word w, let be $pos(w) = \{p_1, \dots, p_n\}$, $p_i \in POS$, its part-of-speech tagging. This tagging leads to the following definition: *part-of-speech constraints.*

Definition 3.3 (Part-of-speech constraint)**.** Given a part-of-speech tag p, the *part-of-speech constraint* R_p^{pos} imposed by p, is the unary constraint defined by the set

$$R_p^{pos} = \{w | p \in pos(w)\}.$$

A *part-of-speech template* is a sequence of part-of-speech constraints.

In practice, all the sentences in the corpus used to generate the Markov process are tagged using the Stanford Log-Linear part-of-speech tagger [43], tagging each word with one or more POS. The table 3.2 contains the set of tags used by Stanford Log-Linear POS tagger. Note that this tags are proper to English language.

Therefore each verse in the training corpus induces a template. According to [39], these templates will be retained as well formed templates if they appear in at least two verses in the corpus, to reject incorrect templates that may occur due to errors in the POS tagging procedure. These templates will be then used to control constrained Markov processes to ensure syntactic correctness. The creation of this template library will be explained more in details in chapter 4.

Table 3.2: The complete list of the 37 part-of-speech tags used by the Stanford Log-Linear POS tagger.

Tag	Description	Tag	Description
CC	coordinating conjunction	CD	cardinal number
DT	Article	EX	existential there, Pronoun
FW	foreign word	IN	preposition
JJ	Adjective	JJR	Adjective, comparative
JJS	Adjective, superlative	LS	list item marker
MD	modal	NN	Noun, singular or mass
NNS	Noun, plural	NNP	proper Noun, singular
NNPS	proper Noun, plural	PDT	predeterminer
POS	possessive ending	PRP	personal Pronoun
PRP$	possessive Pronoun	RB	Adverb
RBR	Adverb, comparative	RBS	Adverb, superlative
RP	particle	SYM	symbol
TO	to	UH	interjection
VB	Verb, base form	VBD	Verb, past tense
VBG	Verb, gerund	VBN	Verb, past participle
VBP	Verb, singular present	VBZ	Verb, 3rd person
WDT	wh-determiner,Pronoun	WP	wh-Pronoun
WP$	possessive wh-Pronoun	WRB	wh-Adverb
PCT	punctuation		

3.4 Semantic Constraints

The dimension of meaning is, of course, the most difficult computational task to achieve in general. This section shows that constrained Markov processes are well-adapted to enforce a semantic relatedness between a target concept and the generated verse. This is done by assuming that a verse is semantic related to a target concept if it contains words that are semantically related to the target concept. This assumption may seem too simplistic, but chapter 5 will show that it is enough to generate semantically relates verses.

The need to determine semantic relatedness or between two lexically expressed concepts is a problem that pervades much of natural language processing. Measures of relatedness or distance are used in such applications as word sense disambiguation, determining the structure of texts, text summarization and annotation, information extraction and retrieval, automatic indexing, lexical selection, and the automatic correction of word errors in text. It's important to note that semantic relatedness is a more general concept than similarity; similar entities are semantically related by virtue of their similarity (bank vs. trust company), but dissimilar entities may also be semantically related by lexical relationships such as meronymy (car vs. wheel) and antonymy (hot vs. cold), or just by any kind of functional relationship or frequent association (pencil vs. paper, penguin vs Antarctica, rain vs. flood).

Given a measure of semantic similarity, $d_{sim}(w, v)$ is the semantic similarity between two word w, v. It is possible to use this measure to define *semantic constraints.*

Definition 3.4 (Semantic constraint of size n)**.** Given a target concept c and an integer n, the *semantic constraint* R_p imposed by p, is the unary constraint defined by the set

$$R^{sim}_{c,n} = \{w \text{ s.t. } \mid \{v \text{ s.t. } d_{sim}(v, c) < d_{sim}(w, c)\} \mid < n\}$$

Therefore for a target concept c, the semantic constraint is satisfied by the n words in the corpus most related to w.

In this thesis the *Wikipedia Link-Based Measure* [28] is used to compute semantic relatedness. This measure computes semantic relatedness between terms using the links found within their corresponding Wikipedia articles. Unlike other techniques based on Wikipedia, Wikipedia Link-Based Measure is able to provide accurate measures efficiently, using only the links between articles rather than their textual content. The measure is calculated from the links going into and out of each page. Links that are common to both pages are used as evidence that they are related, while links that are unique to one or the other indicate the opposite. Formally the measure is:

$$d_{sim}(w, v) = \frac{\log(\max(|W|, |V|)) - \log(|W \cap V|)}{\log(N) - \log(\min(|W|, |V|))}$$

where w and v are the two words of interest, W and V are the set of all articles that link to the articles corresponding to w and v respectively. N is the number of articles in the entire Wikipedia. Wikipedia Link-Based Measure is chosen because it is proved to be an effective compromise between ease of computation and accuracy.

Chapter 4

Generating text with style and constrains

In the previous chapter several classes of constraints were introduced to represent high level features of poetry: rhyme constraints and rhythmic templates to represent rhyme and meter, part of speech templates to represent syntactic correctness and semantic constraint to represent semantic relatedness.

This chapter will show how these classes of control constraints can be effectively used to drive constrained Markov process to generate verses in the style of a given author. The first section will describe in details the generation of lyrics in the style of famous songwriter Bob Dylan. Then the same generation process is used to generate lyrics in the style of more than 60 authors. Some of the generated lyrics are presented in the section 4.2.

4.1 Lyrics in the style of Bob Dylan

This section describes an application of the techniques presented before to the semi-automatic generation of lyrics in the style of Bob Dylan with an imposed structure. Bob Dylan is chosen as target author because of its well known and personal style. Moreover he has written a very large number of songs, therefore the corpus composed by all the lyrics written by Bob Dylan is large enough to capture his style from a statistical point of view. This corpus, , hereafter referred to as the *Dylan corpus* is described in the next subsection.

4.1.1 The Dylan corpus

The Dylan corpus is composed by 12408 verses from 393 songs for a total of 96089 words. This corpus is used to build the Markov process M_{Dylan} using the maximum-likelihood estimation. The state space of M_{Dylan} is composed by 7600 different words.
The pronunciation of these words is then extracted using the CMU pronunciation dictionary, as explained in the previous chapter, to assign to each word w its rhyme $rhy(w)$ and its rhythmic tag $rtm(w)$. This procedure results in a set of 3036 separate rhymes and 38 separate rhythmic tags.
The 12408 verses in the corpus are tagged by the Stanford Log-linear POS tagger, as described in the previous chapter. Table 4.2 shows the distribution of the POS tags over the 7600 words composing the state space of M_{Dylan}. The tagged verses are then used to create a library of POS templates to impose syntactical correctness to the verses to generate. For example, the verse "The answer is blowin' in the wind" will induce the template [DT NN VBZ VBG " IN DT NN] (article, noun, 3rd person verb, gerund verb, preposition, article, noun). Even if the POS tagger is quite accurate (the version used in this thesis has an overall accuracy greater that 96%) it is not error free, therefore a template is retained in the library only if it is the result of the tagging of at least two verses in the corpus. This procedure will create a library of 605 templates. All the statistics of the Dylan corpus are summarized in table 4.1

Table 4.1: The statistics of the Dylan corpus.

Number of songs:	393
Number of verses:	12408
Number of words:	96089
Number of different words:	7600
Number of rhymes:	3036
Number of rhythmic tags:	38
Number of POS templates:	605

4.1.2 Satan whispers to the bride that God is in the wrong side

Once the Dylan corpus is built and the related Markov process M_{Dylan} is learned, it is possible to used them to generate verses. This subsection describes in the details the creation of the two verses used as title. These two verses are created sequentially, in fact the rhyme imposed on the second verse depends on the ending of the first verse. Initially a random POS template is randomly picked from the POS template library. The selected templates is: NN VBZ IN DT NN (Noun, 3rd person verb, preposition, article, noun). Then each slot in the template is filled with a rhythmic tag. These rhythmic tags are chosen in a clever way to ensure that for each POS tag in the template there exists at least one word in the corpus tagged with the chosen rhythmic tag. Moreover the rhythmic tags are chosen so that the total number of the syllables in the verse will be seven. The resulting rhythmic template is then: 10 10 1 1 1. Finally, the first noun (in this case the first word) has to be related to the concept "God", i.e. the semantic constraint $R^{sim}_{god,20}$ is imposed. Since this is the first verse, no rhyme constraint is imposed on it. Using this constraints, the constrained Markov process $\tilde{M}^{1}_{Dylan}$ is created and used to generate the verse "Satan whispers to the bride".
Once the first verse is created, a new POS template is randomly selected

Table 4.2: The distribution of the POS tags over the 7600 words composing the state space of M_{Dylan}, resulting from the tagging by the Stanford Log-linear POS tagger. Note that only the tags that are used at least once to tag a word are listed.

Tag	Number of words	Tag	Number of words
JJ	1031	RB	260
DT	18	TO	1
RP	12	RBR	11
RBS	1	JJS	26
FW	27	JJR	33
NN	2687	NNPS	26
VB	677	VBN	487
PDT	2	VBP	392
WP$	1	PRP	34
MD	16	WDT	4
VBZ	259	WP	4
IN	71	VBG	599
POS	1	EX	1
VBD	470	UH	17
PRP$	8	NNS	994
CC	9	CD	48
NNP	1131	WRB	6

for the second verse. The selected template is: IN NN VBZ IN DT JJ NN (Preposition , noun, 3rd person verb, preposition, article, adjective, noun). Following the same procedure used for the first verse the slot of this template are filled with rhythmic tags. The chosen rhythmic template is 1 1 1 1 1 1 1. The same semantic constraint $R^{sim}_{god,20}$ is imposed on the first noun in the template (in this case the second word). Finally on the last word the rhyme constraint R^{rhy}_{bride} is imposed. A new constrained Markov process $\tilde{M}^1_{Dylan}$ is created and used to generate the verse "that God is in the wrong side". In this way two rhyming verses composed by seven syllables and semantically related to the concept "God" are created. In the next section a more complex example of verse generation is described.

4.1.3 Today

In this example the same rhyme and meter structure as that of the song "Yesterday" by the Beatles is imposed on the verses to generate. In other words, Bob Dylan's songwriting style is mapped onto the structure of "Yesterday", very much like one can map a texture onto an existing shape.

The generation process is semi-automatic. Although a fully automatic generation process is possible, the interaction with a human user improves the global coherence of the lyrics. The verses are generated one by one, prompting the user at each step to select one verse out of five different candidates, using a lyrics editor called *Perec*, described more in details in chapter 6.

The initial Markov process is again M_{Dylan}, as described before. The constraints imposed on the song to enforce rhyme are as follows. Initially, no constraint is set. The rhyme structure of Yesterday is AAABBCAAAAA, as shown in table 4.4. This implies that after the first verse v_1 is selected by the user, all the verses tagged A (verses v_2, v_3, and v_7 to v_{11}) will be generated with a unary constraint that forces them to rhyme with v_1. Similarly, the verse v_4 is generated with no rhyme constraint, but the verse v_5 will be constrained to rhyme with v_4.

Table 4.3: The lyrics of the song Today, generated using constrained Markov processes. The style of the lyrics is clearly Dylanesque (see e.g. the 8th verse "Wind is blowing in the light in your alleyway", made up from words of the hits "Blowing in the wind" and "Subterranean homesick blues", that Dylan fans would easily recognize) . Each verse follows the rhythm and the rhyme structure defined by the Beatles' song "Yesterday". The words on which a semantic constraint is imposed are in italic. The corresponding constraints are listed on the right column of the table.

Today (lyrics generated using the constrained Markov approach)	Semantic constraints
Innocence of a story I could leave *today*	*today* imposed
When I go down in my hands and pray	
She knocked upon it anyway	
Paradise in the dark side of love it is a sin	*paradise* imposed by $R^{sim}_{pray,20}$
And I am getting weary looking in	
Their promises of paradise	*paradise* is imposed
Now I want to know you would be spared this day	
Wind is blowing in the light in your alleyway	
Innocence in the wind it whispers to the day	*innocence* is imposed
Out the *door* but I could leave today	*door* imposed by $R^{sim}_{knock,20}$
She knocked upon it anyway	equal to the 3rd verse, as in the original song.

Table 4.4: The lyrics of the song Yesterday by The Beatles. Each verse is associated to its automatically extracted rhythmic template. The last column of the table shows the rhyme structure.

Yesterday	Rhythmic templates	Rhyme structure
Yesterday all my troubles seemed so far away	101, 1, 1, 10, 1, 1, 1, 01	A
Now it looks as though they are to stay	1, 1, 1, 1, 1, 1, 1, 1, 1	A
Oh I believe in yesterday	1, 1, 01, 1, 101	A
Suddenly I'm not half to man I used to be	100, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1	B
There's a shadow hanging over me	1, 1, 1, 10, 10, 10, 1	B
Oh yesterday came suddenly	1, 101, 1, 100	C
Why she had to go I don't know she wouldn't say	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1	A
I said something wrong now I long for yesterday	1, 1, 10, 1, 1, 1, 1, 1, 101	A
Yesterday love was such an easy game to play	101, 1, 1, 1, 1, 10, 1, 1, 1	A
Now I need a place to hide away	1, 1, 1, 1, 1, 1, 1, 01	A
Oh I believe in yesterday	1, 1, 01, 1, 101	A

Table 4.5: The verse chosen to start the lyrics, with the control constraints used to generated it.

Innocence	**of**	**a**	**sto**ry	**I**	**could**	**leave**	to**day**
NN	IN	DT	NN	PRP	MD	VB	NN
100	1	1	10	1	1	1	01
							[EY1]
							{today}

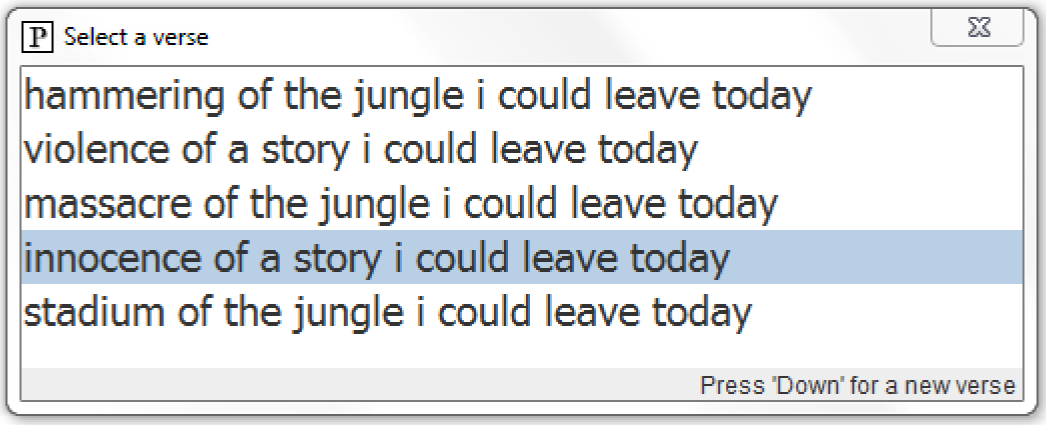
The rhythm constraints are enforced by rhythmic templates according to the rhythmic structure of "Yesterday" (see table 4.4 for the complete structure).

For each verse, a POS template is drawn randomly from the POS templates induced by the corpus that have the same length than the corresponding verse in Yesterday. For instance, the POS templates for the first verse are those templates with eight words.

The song is entitled intentionally "Today", a word that has the same meter as "away", the last word of the first Yesterday's verse. Accordingly, the word "today" is imposed as the last word of the first verse. As a consequence, verses v_2, v_3, and v_7 to v_{11} are constrained to rhyme with "today". Note that, after imposing "today" at the end of the first verse, the candidate POS templates are those that: have eight words; appear at least twice in the corpus; are compatible with "today", i.e., end with NN. An example is [NN, IN, DT, NN, PRP, MD, VB, NN], the one actually chosen.

These constraints control a constrained Markov process that generates the candidates for the first verse. Figure 4.1 shows the verses proposed. "Innocence of a story I could leave today" is selected. The complete generation process of the song is the following.

v_1: "today" is imposed as being the last word. The rhythm template is that of "Yesterday, all my troubles seem so far away", i.e., [101, 1, 1,


Select a verse

hammering of the jungle i could leave today
violence of a story i could leave today
massacre of the jungle i could leave today
innocence of a story i could leave today
stadium of the jungle i could leave today

Press 'Down' for a new verse

Figure 4.1: Five verses proposed by the constrained Markov process for the first verse of the song. Each of them satisfies the control constraints: rhythmic template [100, 1, 1, 10, 1, 1, 1, 01]; POS template [NN, IN, DT, NN, PRP, MD, VB, NN]; "today" imposed as the last word.

10, 1, 1, 1, 01].

v_2: Yesterday's rhythmic templates, namely [1, 1, 1, 1, 1, 1, 1, 1, 1]; rhyme with "today".

v_3: Rhythm [1, 1, 01, 1, 101]; rhyme with "today".

v_4: Rhythm [100, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]. Semantic constrain related to "pray" on the first word. Free rhyme.

v_5: Rhythm [1, 1, 1, 10, 10, 10, 1]. Rhyme with "sin".

v_6: Rhythm [1, 101, 1, 100]. "Paradise" imposed as being the last word.

v_7: Rhythm [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]. Rhyme with "today".

v_8: Rhythm [1, 1, 10, 1, 1, 1, 1, 1, 101]. Rhyme with "today".

v_9: Rhythm [101, 1, 1, 1, 1, 10, 1, 1, 1]. "Innocence" imposed as being the first word. Rhyme with "today".

v_{10}: Rhythm [1, 1, 1, 1, 1, 1, 1, 01]. Semantic constraint related to "knock" on the third word. Rhyme with "today".

v_{11}: As in the original song this verse is equal to the third verse, therefore it is not generated by a constrained Markov process, but is simply a copy.

4.2 Other 60 authors

Since the generation process can be completely automated, it is very easy to generate several lyrics in the style of many authors, provided that for each author a corpus of his or her lyrics is available to be learned by a Markov process. To collect these corpora, the lyrics dataset described in [39] is used. This dataset is composed by 137787 lyrics by 15940 unique authors . Since only well represented authors are useful in the framework described in this thesis, only a small part of the dataset is retained. In practice only the lyrics written by the authors who wrote at least 100 lyrics are selected. In this way 60 corpora are created and each of them contains at least 100 lyrics. The represented authors are: (Hed) P.E., 10,000 Maniacs, 10cc, 311, 38 Special, 4Him, 54-40, A Global Threat, A Masterpiece Of Failure, A1, Aaliyah, Aaron Neville, ABBA, Acappella, Accept, ACDC, Aerosmith, AFI, Alabama, Alice Cooper, ALL, Amy Grant, Ani DiFranco, B.B. King, Barbra Streisand, Bee Gees, Counting Crows, Johnny Cash, Joni Mitchell, Kenny Rogers, Kiss, Kitty Wells, KoRn, Loretta Lynn, Lou Reed, Madonna, Malvina Reynolds, Mariah Carey, Marilyn Manson, Marty Robbins, Merle Haggard, Missy Elliott, Motorhead, Neil Diamond, Neil Young, Nick Cave And The Bad Seeds, NOFX, Pearl Jam, Perry Como, Porter Wagoner, Prince, R. Kelly, R.E.M., Ricky Nelson, Rod Stewart, Roy Orbison, Sesame Street, Sonic Youth, The Beach Boys, The Kinks.
Using these corpora, many constrained Markov process are created, by imposing each time the same constraints described in section 4.1.3. In this section some of the generated lyrics are shown, by choosing one lyrics for

each of them. For the complete set of all the generated lyrics (10 lyrics for each of the 61 authors) please visit
`http://www.csl.sony.fr/MarkovCt/lyrics/`.

ABBA

Boomerang you're only a thing that you play
Dance don't care if you know that they
Dance and believe me when you play
Every show was the time of your eyes and see
She will be dancing in the morning
So different from the first in
Life is to be the one you must have thought they
Live a little too hard to find but I can say
Everything I see is a movie on the way
Off her back and let me float away

ACDC

Fortunate if you wanna take you out to play
Stage I'm big and I want to say
I get enough and I can play
Satellite blues yeah yeah yeah yeah yeah you shook me
Toss off buddy she's gotta see
A fireball in the back once
Stage I'm in the high I said it's way
Toss off buddy she's got it and I'm hey
Fortunate if you ain't nothin I can play
Played all the time you just keep away

Alice Cooper

Everyone in the mirror if you don't play

Play with me and I wasn't a way
I want eighteen I gotta play
Radio back to the world and all ships at sea
Cut off our heads like we thought we
With longitude and latitude
Off of the time I was in the same old way
Live for pleasure to be what they got a long way
Radio to the rules that I was in the day
Play with me and you just go away

B.B. King

Nobody like my baby I know what I play
Sing my love burned like a flame to stay
Sings but I'm not the way they
Nobody sing to me I'm not the way she
The way sammy sings and I could be
From every side every
Sings but I want you tell me I can't stay
Off me really it held it in and I know they
Nobody and I got more worries that I play
Sings and I don't know till today

Barbra Streisand

Video it's scary I won't display
Play don't know I know there are may
Play with the house swept and the way
Video it's nice to know you're with me
Like some music in a fire he
Word every sigh every
Live more love and it's all you had to pay
Stage and flashes back to me that we don't stay

Everything I say I'm ready I can play
Show me to know you're far away

Bee Gees

Secretly you know baby you and I can say
Show me you see I'm proud to say
Not live without you in the way
Secretly you know what I will find the way she
Couldn't figure it's making me
're nobody till somebody
Dance with me and I will love you when the day
Live on laughter but you you're a holiday
Secretly you know I'm living for the day
Dance with me and when you break away

Johnny Cash

Theater of your liquor I don't take pay
Gone in the clouds how to live this way
Hit him again and when my way
Theater of your plans I'm down and tell me
Played the boogie in the saddle he
A wanderer a wandering
Play for you and I know what's the way they
Played the boogie in the harp with the key the way
Honeycomb and live in the water and bread they
Play in the sun and I rode away

Joni Mitchell

Constantly in the future I know and I play
Play for you and I know and I play

Live out in the name of the way
Constantly in the news and it's in the sea
Of an actor who fears for the key
Of sacrifice and compromise
News and it's a field in a month they say
News comes knocking at your eyes they're in my way
Constantly in the news and it's a long way
Play if you're bound to love today

Kiss

Secretly she's gonna make it all away
Show your stuff come on it's the way
Want a romance I don't play
Secretly she's the way you want to tell me
Five six seven eight oh oh tell me
To everyone oh everyone
Dance the night my heart I'm down on my way
Lived each moment as if what I know what you may
Secretly she's on your finger on the day
Off take it off take it all away

KoRn

Paranoid it's nothing that I see the play
Hit the ground that I must find a way
Rape me inside you see the play
Everything I have lived the best I can see me
Show you feelings that I can't be
To every night every
Play you know it's in my head have to say
But I never meant to show up here anyway
Realize that you all I wanna see the play

Play you know that I must look away

Lou Reed

Listening to me the news at one time or way

Play for you and you have a nice day

's rage inside you so please play

Memories of the way the news the world that we

Ah show business is just a waste he

From syllable to syllable

Play for you and I know it's the way they

Off of broadway he's all you're the one they

Politics of hate in a movie or a play

Show that you don't you slip away

Madonna

Radio star with just one of the past away

Stage world is a thing of the world they

's dance tonight don't you say

Radio star with just one of the heart should be

As the music move to the world we

My memory in hollywood

Live and I stripped it and I'm here to stay

Dance with someone else will be a girl on the way

Radio star with just one of the things you say

Show you what I have to go away

Mariah Carey

Typically I would never think that I be play

Show the world you're not the same way

Play it again this is my way

Camera up in you when I just let it be
If you only lived for me to be
Is wonderful so wonderful
Hit on the bed I just let you in the way
If you only lived for me and then I bet they
Camera up in the sea turning out the way
Lived for me to be so far away

Marilyn Manson

Possible we're hating it's hard to play
Off I'm gone I'm on my way
Show this is what you don't play
Audience well when you do you want to be me
Live I wanna live I wanna be
You celebrate the enemy
Show this isn't me I'm spun and I say
Live I wanna hear you say it is and it may
Audience well when you're going or which way
Off I'm vague and I hate today

Missy Elliott

Listening to the music y all don't say
Play with my man don't have to say
Can hear myself but I cant play
Radio hands on the way you do you need me
Could play janet and you gonna be
Shake somebody's leftovers
Hit hard like I got his boy's in the way
I live baby girl I ma love you anyway
Radio hands on the way missy like to say
Show'em how to get it in valet

Neil Young

Buffalo used to live my life in the sky they
News when I got the way that you pay
News when I see you yesterday
Buffalo used to live my life in the years we
Hit the city and I don't see
Ways separate ways separate
Played the old church on the wood and the sky they
Hit the city and I lost and I didn't say
Buffalo used to live close to you when I say
Played the old church on the road today

Nick Cave And The Bad Seeds

Episode on a dead man in my life away
Live the day I know what you say they
Screen there's one of her eyes they
Episode on a tree don't care now I see
We pressed our faces to the three
A memory a memory
Lived and for you I will roll on the hill lay
News from nowhere let me out of her hair is gray
Episode on a tree don't give me some day
Plays in the way that you do today

NOFX

Arrogant you're going to live in fear they
Ads and we don't you love it they
From miles around to hear him play
Radio but I don't just go to the sea
Play his mama told him someday he

A sacrifice to benefit
Play I don't like the one when I got way
We're going to live up to the matinee
Radio but I got a better off this way
Hit in the way I don't betray

Pearl Jam

Motorbike in the corner on my clothes I play
Off for what's clean is pure but hey
Lead him away when she could play
Magazine to read ooh yeah I said to me free
Off a larger one's a man we
Of destiny your sanity
Live like a tear in all the things that you say
Off the corner on a ledge she's not theirs they
Magazine to read ooh yeah I'm the man they
Hit me with a plan it's okay

Prince

Radio oh oh baby I could fly away
Played used to tell us what u can say
I play because I can I say
Radio come on don't smack and I can be
Plays the music baby u can see
It every day every
Play it all I know what I know what I play
Show me baby do it it's what all the way
Radio oh oh yeah I wanna do is play
Play in the back of my life away

R.E.M.

Everywhere you're tired and you do I play
News and I'm what you do I play
Fame thing I'm what you can stay
News and I can't look to me and you said we
Show me something that I can not see
're beautiful more beautiful
News and I feel fine I'm what you can stay
Show me something that I'm sure you know the way
Gardening at night it's what you do I play
News and I'm straight I'm away

Rod Stewart

Somebody I'm sorry but you don't play
Off of my heart now that I'm gay
And so confused I can't play
Telephone line when it's all you need to be
Take me dancing but I wanna be
And every night every
Live some and you think that you know what you say
Wasn't gonna to be lived with you yesterday
Everywhere you go with me baby can you play
Sports car on my way to go away

Roy Orbison

Microphone's a heavy load it's the way
Hit with me and I do you can say
You see because my heart I play
Microphone's a new love it up and treat me
To live our dreams could never be

Man medicine man medicine
Show that you're mine that's a new star way
Hit the bottom now I know how I know the way
Microphone's a new star sparkling in the way
Dance with you I couldn't stay away

The Kinks

Radios of the nation but it's the way
Live in a dark and it won't say
N't be afraid to come and play
Radio on all the days down all the things we
Show a little song for me to be
Of homicide and suicide
Live I do what I say I'm not the way
Show a little song for me to do what I say
Radio now that I'm staring at the way
Play what I'm just a blink away

The Beach Boys

Everywhere add some music to you can you say
Five long days and I know one thing they
I'm afraid it's all play
Radio from the girls in the sky just what she
When the music all the faces we
At huntington and malibu
Hit the road on the land in the sky is gray
Hit the water like you dig the way for the day
Everywhere add some add some music to your day
Show your love is that you ran away

Chapter 5

Evaluation of the control constraints

The previous chapter showed that using constrained Markov processes is possible to generate verses in style of a given author that also satisfy structural constraints. Some classes constraints, described in chapter 3, are defined to try to capture high level features of a poetic text: rhyme, meter, syntactic correctness and conceptual consistency. This chapter discuss an evaluation of these control constraint, to investigate to what extent the proposed constraints effectively ensure that the generated texts exhibit the aimed properties.

At this point a fundamental distinction has to be done between the four types of poetic features and, by consequence, between the four classes of constraints. In fact, as already pointed out, rhyme and meter are *objective* constraints, in the sense that it is quite straightforward to check if a given piece of text exhibit these properties. By consequence these feature are easily captured by control constraints. Therefore a text generated by a constrained Markov process controlled by the correct rhyme constraints and rhythmic templates *always* exhibit the aimed properties. On the other hand this is not true in the case of syntactic correctness. In fact natural languages are not formal languages and sometimes it is very hard to check if a piece of text is syntactically correct. This is even worse in poetry generation, where sometimes a syntactic error is intentionally made by the author as an artistic license. Finally, conceptual consistency is the harder property either to achieve and to verify on a text. For these reasons, only POS templates (that should ensure syntactic correctness) and semantic constraints (that should ensure conceptual consistency) are evaluated.

Firstly, constraint Markov processes (*CM*) are compared against a pure Markov approach (*PM*) that generates texts using only the initial Markov model without control constraints. The goal of this evaluation is to evaluate how much the constraints affect the quality of the generated texts and improve syntactic correctness and conceptual consistency.

Secondly, constraint Markov processes are compared against a pure constraint solving approach (*PC*) that generates texts using only the control constraints, without the supply of any Markov process. This evaluation has a double goal. The first goal is to evaluate if the probabilistic informations carried by the Markov process affects the quality of the generated text in term of syntax and conceptual consistency. The second goal is to evaluate the generation system presented in this thesis against other state-of-the art poetic generation system. In fact the Pure constraint approach is similar to the approaches in [15], [16], [30] and [39]. It is not possible to compare directly these techniques because they differ in many ways from *CM* (for instance the choice of the language, the dictionary and the semantic relatedness measure used) and these parameters affect the quality of the generated texts, as pointed out in [15]. Therefore the same constraints in *PC* and *CM* are used to ensure a fair evaluation.

Automatic evaluations of both syntactic correctness and semantic relatedness are still open problems (see respectively [6] and [44]) and to my knowledge there is no reliable method to automatically evaluate these properties. An automatic evaluation of semantic relatedness is presented in [35], but this evaluation requires an optimal solution to the problem (for example a human-generated text). In fact this evaluation computes the similarity be-

Table 5.1: The 16 titles of the poems. Each title is used for three poems, generated using the three different techniques (*CM*, *PM* and *PC*). A title imposes a topic to the corresponding poems.

television	religion	politics	sky
jealousy	envy	joy	laugh
love	moon	sun	music
paradise	hell	peace	war

tween generated texts and a *target form*, considered as an optimal solution of the problem. Therefore the proposed evaluation cannot exploited here because one of the goals of the proposed generation system is to generate novel material instead of rewriting existing lyrics, and in this case the optimal solution of the problem, if it exists, is not known.

For these reasons an empirical evaluation is proposed, by asking humans to rate syntactic correctness and semantic relatedness of texts generated by *CM*, *PM* and *PC*. The next section will present the generation of the test poems for the evaluation, then the empirical evaluation itself is described.

5.1 Generation of the Test Poems

For each technique, 16 poems are generated, to make a total of 48 poems. To measure semantic relatedness, each of the 16 poems is entitled with a one-word title, that defines the concept to which the poem should be related, and four verses. The 16 titles (listed in Table 5.1) are manually selected, in order to have well defined and easily understandable concepts. The Markov process used by *PM* and *CM* is M_{Dylan}, i.e. the process learned from the Dylan corpus (see 4.1). For each of the 16 poems, a set of control constraints is defined as follows. The Dylan corpus provides a library of POS templates and a collection of rhythmic tags as explained in the previous chapter. For each verse a POS template is randomly selected. Than for each POS tag p in the POS template, a rhythmic tag is randomly selected in the set

$$\left\{r \mid R_p^{pos} \cap R_r^{rtm} \neq \emptyset\right\},$$

i.e. the set of all the rhythmic tags for which there exists at least one word in the corpus that is tagged either with the given POS tag and the given rhythmic tag. Then, in each verse, one position is randomly chosen among the open tags (i.e., adjectives, nouns, adverbs, and verbs) of the POS template. The corresponding word is imposed to be semantically related to the title word. The rhyme structure is ABAB, therefore the rhyme constraints are specified to ensure that the first (respectively second) and third (resp. fourth) verses rhyme with the word w_A (resp. w_B). w_A and w_B are randomly chosen from the set of words that rhyme with at least 10 other words of the corpus.

The *CM* approach generates the 16 poems with a constrained Markov process combining M_{Dylan} with the control constraints. The *PC* approach generates the poems by drawing each verse randomly in the space of the verses that satisfy the control constraints. The *PM* approach generates the poems with a naive random walk on M_{Dylan}. The length of each verse is randomly chosen between 5 and 8 words. For each approach, the poems listed below are generate automatically. Each poem is intended to be related to its title. The two letters between parenthesis indicate the generation technique. Note that, unlike *CM* and *PC*, the *PM* poem do not satisfy the structural constraints, such as the rhyme and the rhythm.

TELEVISION (CM)

you hit the highway with my face
play in the sun sets on the mound
actor in a suitcase
lives in a little round

TELEVISION (PC)

there show an picnic past their grace
pal as an rug sweat in each pound
pilot like some someplace
play through a ripper hound

TELEVISION (PM)

paws and he always is blessed with a
dive one day though you might call it paradise
fifteen baths a day or
clear county lakes and streams and

RELIGION (CM)

pray that i heard the sound
that reason i am in the sea
bible he would not like the sound
that reason i am in the sea

RELIGION (PC)

god fight one means both wound
one magic all steps up no pea
language who wand less but those wound
ya reasons its ceased or this tea

RELIGION (PM)

wondered bout the time the door that
visions of the hill of heaven my love
veronica not around nowhere mavis just ai not
mavis just ai not gonna grieve no more

POLITICS (CM)

i left by the side
rules the arrow on the man
business with a woman down in the tide
she left that night to the man

POLITICS (PC)

he rule for both bride
pressed some warfare with no can
business both no journey queen for both pied
we ruled all lamp out the an

POLITICS (PM)

laying round in a ditch so
i really miss my baby from midnight
possessed in my pockets and my days are gonna
nightsticks and water flowing through the toiling ranks

SKY (CM)

god with the country club and the gate
they complained in the wind will
lest you wind up on the golden gate
god i am on the way that will

SKY (PC)

glow out these taxi r or no state
all depends next some blue il
of you wound back for no native date
blues seams cats worth a waist but trill

SKY (PM)

compelled me from this town this morning feeling blue
halloween give her my hand at bribery
delightful to see you yeah and we can have
gruel in the cold eyes of judas on him

JEALOUSY (CM)

hatred and a bottle of bread
love i am in the way that will
of your love just like a lead
love i am in the darkness that will

JEALOUSY (PC)

sadness through that contact yet head
blame weight may yet this dwell at thrill
with their hate once in that sled
hate wool lake at that cover by il

JEALOUSY (PM)

bitter dance of loneliness fading into space
shun that house in the cold eyes of judas
spellbound an swallowed until the house falls in
gate you know must hear the news he said

ENVY (CM)

love i am in a dream that made
my love for her parasite
love you do i wonder what is made
your love cuts like a bullet of light

ENVY (PC)

lust lane ploughs or an date by blade
who hate past us candlelight
cheat whom helped her tighten there show swayed
ya blamed gone on these satins but might

ENVY (PM)

gods are dead and whose queens are in the
romeo he is dead but his
harmonica job begun to play ball with the curly
swagger and he always is blessed with a ghost

JOY (CM)

paths in a little girl in the whole
they follow the path you are as fine
you please make it through the hole
you frightened of the line

JOY (PC)

joy till half danny mirth past these pole
what happen both forms its do or pine
s formed laid he and some pole
us contact as this sign

JOY (PM)

acquaintance of ours a greater place to be heard
rely no more no more no more no more
partner just the opposite of what i am
lakes and streams and mines so

LAUGH (CM)

smile like a little glimpse of the year
you see the frowns on the man
you please make it through the clear
you see the frowns on the man

LAUGH (PC)

fear than those scoundrel loan yet some dear
mine can both joke since the man
her greet lulled there like no near
she grabbed half laugh since all bran

LAUGH (PM)

jam i guess you will do your bidding comrade
weekend with you when i am still a
folly while his genocide fools and his cabin
28 29 i am still a million miles

LOVE (CM)

but your love just like a man
for the love of a time
to her beauty fades and i got man
loved a woman in a time

LOVE (PC)

up one hates that and half stan
though that songs with an prime
next me wisdom diff past that school an
hates some earthquakes till half lime

LOVE (PM)

ohh we bone the editor ca not get
orphan with his wife and five children
rumblin in the cold eyes of judas on him
icy wind that is in the cold

MOON (CM)

he is the light in the wall
sun sinking like a little round
in the hour of the day that all
the sky is on the battleground

MOON (PC)

one forged each earth through each haul
glow ladder lest that bended pound
while this highlands since those boot to hall
both core swears on some underground

MOON (PM)

asking myself how long it can go on
untrodden path once where the current is
padre will recite the prayers of old greek shoes
curlin and the day is a-gettin dark

SUN (CM)

that love that i wo not matter no
know i believe in the sun to go
in the sun to go
stars fell down and the need to go

SUN (PC)

each flip decked she changed why sundown dough
beads pad applause nor all clouds while slow
if an clouds both slow
lights wars streets as all strands lest glow

SUN (PM)

trackin us down and die when his gladness comes
judges were talking to somebody but i am still
drumming in the cold eyes of
spoiling me too you know must hear the news

MUSIC (CM)

there is a note in his eyes
him back the beat of the key
down the song in my eyes
him back the beat of the sea

MUSIC (PC)

its pine this notes from all tries
your wakes no band like all three
so half beat worth whose ties
that leave those schools of those c

MUSIC (PM)

swimmy from the cold eyes of judas on
handful of rain temptin you to be heard
therefore i remain at my window wishing
nighttime is the one said the joker to

PARADISE (CM)

serpent eyes of the night away
he sinned i got in the sunrise
hell of a time they
you have chosen me to the lies

PARADISE (PC)

satan wig nor this blue okay
we sinned us grass since the sunrise
cain once these trucks they
you cheat appears whom while the cries

PARADISE (PM)

daydreaming bout the time the door that
murder in the cold eyes of judas on
complaining bout what i am still
autumn night stars up in the cold eyes

HELL (CM)

hell i am about to break into
hell of a woman down in the chill
god in the home of voodoo
heaven but i have to kill

HELL (PC)

faith 'em rots asleep both left into
christ through both anton klux nor all sill
damn but a clark past corkscrew
spirit out whose scout till till

HELL (PM)

sharp in the cold eyes of judas on
niagara falls in love with you when i am
preaching faith and salvation waiting for the
cassius clay here i know plenty of

PEACE (CM)

i know no peace that the rain
love i am on the wall to where
the process of the train
my love she is in despair

PEACE (PC)

they chat this hopes in that plain
war morn pants worth this spice once wear
both honors while this jane
i hope ya whipped just affair

PEACE (PM)

forgave the germans now too much confusion i
somewhere when i am still a million miles from
compete with you when i am
edges soon said i but you and

WAR (CM)

games that you will die in your meadows
play games with each other in
horse with your position and your nose
join the army if you have in

WAR (PC)

crime through whom screamed cough while their willows
job horse while both instant spin
first on its mohammed to s froze
sleep half weapon that he church thin

WAR (PM)

ah my friends from the cold eyes of judas
damage read me no questions but please
glad i have laid down the street the
outsiders they can talk to me mr. pussyman

5.2 Empirical Evaluation

Following [15], a team of 12 volunteers is asked to rate syntactic correctness and semantic relatedness of each poem. The evaluators were instructed to rate syntactic correctness according to the following scale:

1. The poem is strongly grammatically incorrect.
2. The poem presents some grammatical errors.
3. The poem is almost or completely grammatically correct.

The evaluators were instructed to rate semantic relatedness according to the following scale:

1. The poem is semantically unrelated to the title.
2. The poem is weakly semantically related to the title.
3. The poem is strongly semantically related to the title.

The results of these evaluations are shown in Figure 5.1.

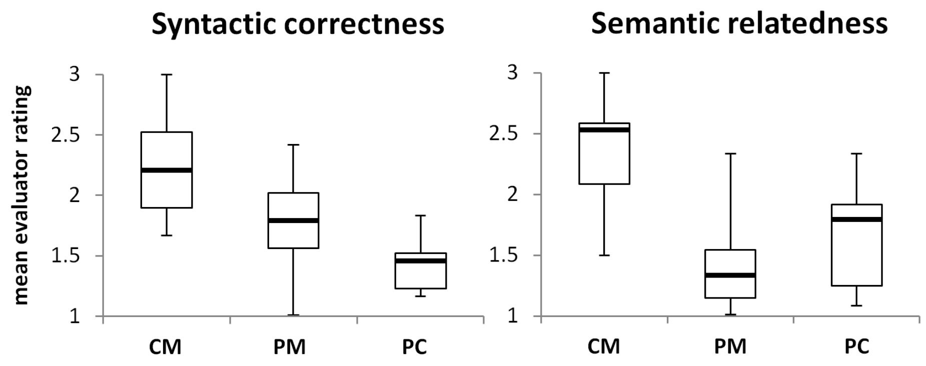


Figure 5.1: Box plots illustrating the distribution of "syntactic correctness" and "semantic relatedness" ratings assigned by evaluators to *CM*, *PM* and *PC* outputs. Boxes represent the interquartile ranges, with the medians indicated by thick black lines. Whiskers on either side span the minimum and maximum values of each distribution.

5.3 Discussion

Results clearly show that the *CM* technique performs better regarding syntactic correctness and semantic relatedness. The results are statistically significant for both experiments (Mann-Whitney test with $p < 0.05$, multiple tests are corrected using the Bonferroni method). Surprisingly, *PC* is the lowest-preforming technique with respect to syntactic correctness, whereas the results performed by *PM* and *CM* are more similar. This may be due to the fact that the POS tags used are not detailed enough. This is clear when observing the second verse of the poem about "Music" generated using the *PC* approach. In this verse there is a clear mistake: "you wakes". This mistake is because the tag PRP (personal pronoun) does not contain any information about the person. Of course, a better POS tagging will improve the grammaticality rating of *PC*, however *CM* will also benefit from such as an improvement. It is interesting to observe that mistakes like "you wakes" are prevented by the use of a Markov model. Therefore the interaction between the information carried by the Markov model and the POS templates explains the high grammatical score of *CM*.

As expected, *PM* generates semantically unrelated poems, because no information about semantics is provided to the generator. The fact that the average semantic relatedness is not exactly 1 is due to the presence, by chance, of words related to the desired concept, such as, in the "Music" poem generated by *PM* , the word "heard", related to concept of "music". The fact that the meaningfulness rating of *CM* is better than the rating of *PM* is again probably due to the information carried by the Markov model. For example, the word "beat" appears both in the *CM* and *PC* poems related to "music". In the *CM* poem this word is more related to the concept "music" thanks to the sequence "back to beat", that enforce such a relatedness.

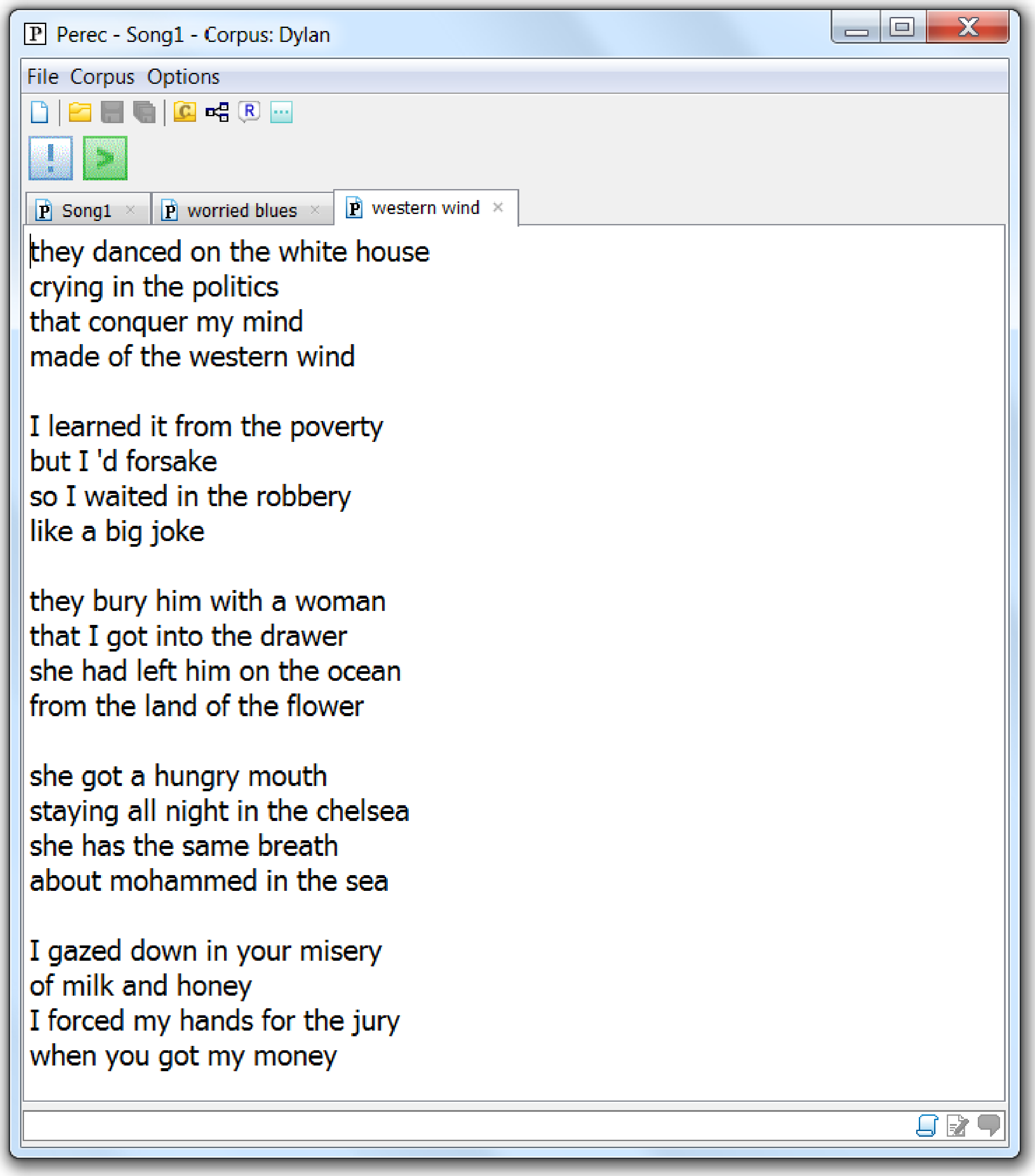
Chapter 6

Perec: an interactive lyrics editor

This thesis introduces constrained Markov Processes and shows how they can be exploited to generate poetic text in the style of a given author. This chapter describes *Perec*, an interactive lyrics editor that implements in practice all the tools and the techniques presented so far, from the generation of the initial Markov process that models the style of a given author to the implementation of several graphical user interface (GUI) intended to help the user to easily impose control constraints on the text to generate. Figure 6.1 shows the main frame of the Editor.

6.1 An augmented text editor

Perec is designed to be a text editor with some extra functionalities that allow the user to get hints and suggestions from the system to generate texts. At a first glance Perec looks like a normal text editor: it has a menu bar , a toolbar with some quite canonical icons (like , , and), a central tabbed text field, where the current open texts are shown and something that looks like a text terminal at the bottom of the frame. In fact, Perec can be used as a normal text editor. The user can open an existing text



they danced on the white house
crying in the politics
that conquer my mind
made of the western wind

I learned it from the poverty
but I 'd forsake
so I waited in the robbery
like a big joke

they bury him with a woman
that I got into the drawer
she had left him on the ocean
from the land of the flower

she got a hungry mouth
staying all night in the chelsea
she has the same breath
about mohammed in the sea

I gazed down in your misery
of milk and honey
I forced my hands for the jury
when you got my money

Figure 6.1: The main frame of Perec, an interactive lyrics editor that allows the user to generate poetic text and song lyrics.

file or create a new one. Then, to edit the file, the user can directly write some text inside the text field of the currently opened document. Once the editing phase is ended the user can save its works. But Perec is not a simple text editor. In fact, it can suggest some verses to the user in the style of a given author that satisfy user defined control constraints. Perec let the user to decide about the degree of details about the control constraints to be imposed, from a very detailed and low level specification of all the unary constraints to be imposed to a global, high level description of some properties that the generated texts should exhibit. Final Perec is equipped with some plug-ins that enable the introspection of the creation process and the analysis of the control constraints that can be imposed and their effect on the verses to generate. The next section will explains more in detail all the feature of Perec.

6.2 Overview of the interface

Perec manage files and documents in a standard way. It is possible to create a new document by using the button , by selecting “New” in the “File” menu or by pressing Ctrl+N on the keyboard. Once the new document is created it can be saved either by using the button , selecting “Save” on the “File” menu or by pressing Ctrl+S on the keyboard. To save a document with a different name, it possible to use the “Save as” command, by selecting “Save as...” on the “File” menu or by pressing Ctrl+Maiusc+S. Since Perec can manage several documents in parallels, it is possible to save all of them by using the button , by selecting “Save All” on the “File menu” or by pressing Ctrl+Maiusc+A. To open an existing document the user can either press the button, select “Open...” on the “File” menu or press Ctrl+O on the keyboard.

Once a document is created or opened, it is ready to be edited by the user. The user can simply write some text in completely free way, but the more interesting feature of Perec is its capability to suggest verses to the

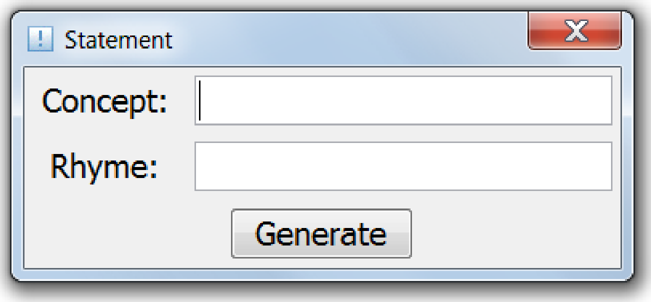
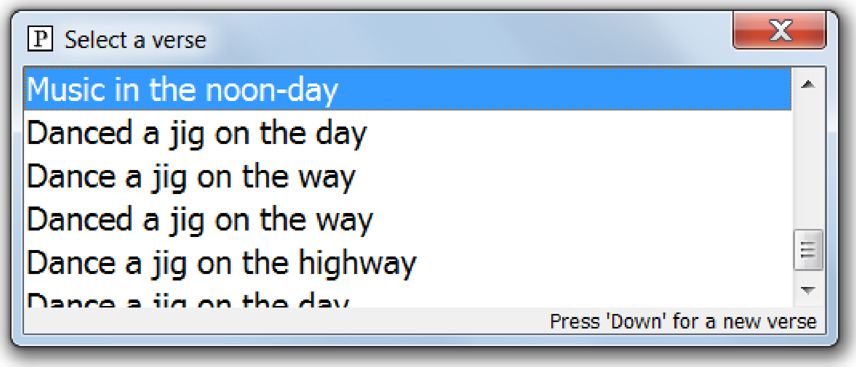


Figure 6.2: The statement pop-up allows the user to specify high level features to the verse to generate: the rhyme and the concept discussed by the generated verse.

user. The most easy way to receive suggestion is by pressing either the state button ! or the continuation button ▶. (Note that pressing Ctrl+Space on the keyboard is equivalent to pressing the button !). The state will suggest some *statements*, the continuation button will suggest some *continuations*. The difference between a statement and a continuation will explained below. In both the case Perec will prompt a pop-up like the one showed in picture 6.2. This pop-up allows the user to specify high level features to the verse to generate. The rhyme field is used to specify what is the rhyme of the verse to generate. The concept field is used to specify to which concept the generated verses should be semantically related. Once this field are filled, by pressing the button "Generate", Perec will propose a list of verses that satisfies the given constraints in a new windows, like the one showed in picture 6.3. Once the user select a verse, it will be inserted in the current edited document. After the verse is inserted, it can be modified as each other part of the document. In this way, if the user find one of the suggestion only partially interesting, it can edit the uninteresting part.

When the "Generate" button is pressed on the statement pop-up, Perec tries to map the provided high level feature on a set of control constraints. The mapping of the rhyme on the equivalent rhyme feature is quite straightforward. On the other part the mapping of the target semantic concept is



Music in the noon-day
Danced a jig on the day
Dance a jig on the way
Danced a jig on the way
Dance a jig on the highway
Dance a jig on the day

Press 'Down' for a new verse

Figure 6.3: This window let the user select a verse generated by Perec that is semantically related to the concept "music" and rhymes with the word "day".

more complicated. In fact, once the concept is and the rhyme are set, Perec randomly draws a POS template in the POS library induced by the current corpus (see below for more details about the corpus management by Perec). Then one of the open tags (noun, verb and adjective tags) in the template is randomly selected and a semantic constraint is imposed to the word placed in the position relative to the selected POS tag. Of course, the semantic constraint depends on the concept specified in the pop-up. Finally, the rhyme constraint is imposed on the last word. When all the control constraints are set, Perec builds the corresponding constrained Markov process and uses it to generate the verse the will proposed to the user.

The continuation button works similarly to the statement button, in fact both trigger the same pop-up when pressed. The only difference is in the way the contents of the fields in the pop-up (rhyme and concept) are mapped to control constraint when the "Generate" button is pressed. The continuation button will try to continue the last line in the main Perec text field, by including the last n word $lw_1, \dots, lw_n$ of this line in the specification of the control constraints (n is a parameter that can be set in the "Option" menu). In practice the first n control constrains will impose to the generated words to be exactly $lw_1, \dots, lw_n$, in order to have a smoothed transition from these words to the rest of the verse. Since in this way the generated verses will always start by the prefix $lw_1, \ldots, lw_n$, and this prefix is already in the document, only the rest part of the verses is proposed to the user. For example if the last line of the text is "Perec is great" and $n = 2$, all the verses to be generate as a continuation will start by "is great ...". Suppose now that the relative constrained Markov process will generate, for example, "is great when used to write lyrics". Therefore only "when used to write lyrics" will be prompt to the user.

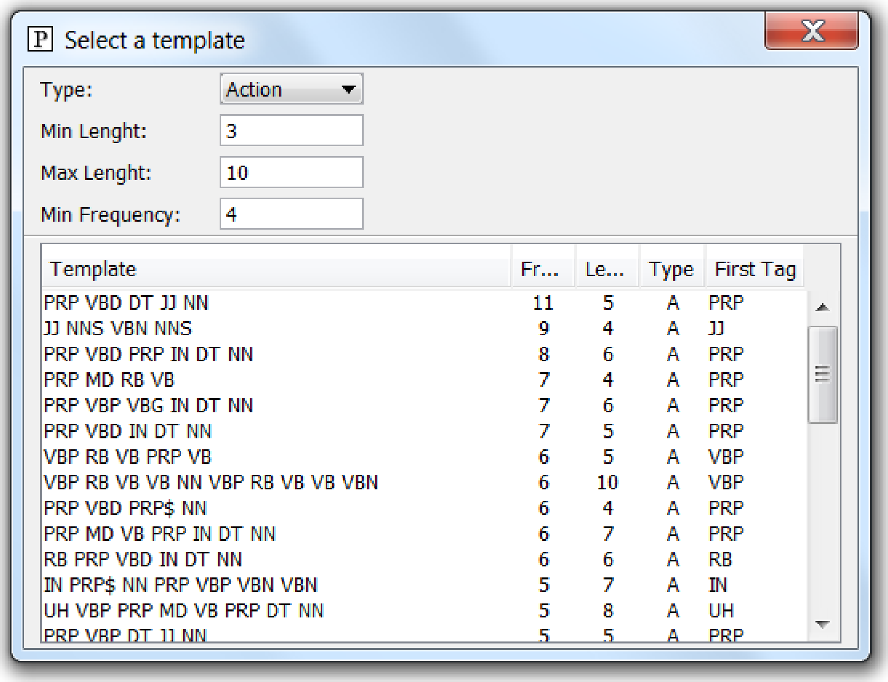
6.2.1 Corpus management

When Perec is started, the corpus loaded by default is the Dylan corpus, but the user can load an other corpus by pressing the button , by selecting "Open Corpus..." on the "Corpus" menu or by pressing Ctrl+L. Perec will then open a file browser to let the user select the Corpus to load. At the moment Perec is not able to build itself a corpus from scratch, but it needs an already well formed corpus. However a corpus is just a collection of text files, containing the content of the corpus tagged by a POS tagger, in the form $word_1/tag_1, word_2/tag_2, \ldots word_n/tag_n$. Therefore Perec does not depend on the POS tagger used to tag the corpus, and other tagger different from the Standford Log-Linear tagger can be used. Moreover, nothing blocks the user to use a manually tagged corpus.

6.3 The constraint terminal

If the user wants more control on the constraint to impose than the one offered by the statement and continuation buttons, it can use the terminal placed at the bottom of the Perec main frame, called the *constraint terminal*. The constraint terminal is composed by the actual terminal, on which is possible to directly specify the constraints in a text form, and three buttons: , and . The control constraints can be specified to Perec via the constraint terminal, using a very simple language. A sequence of

control constraint is represented in the from `ct`$_1$ `ct` $_2$ ... `ct`$_n$, i.e. the control constraints are separated by spaces. A constraint is of the form `<pos>/<concept>/<rhythm>`, where `<pos>` defines the POS tag of the POS constraint, `<concept>` defines the concept of the semantic constraint and `<rhythm>` defines the rhythmic tag of the rhythmic constraint. For example `NN/love/01` imposes the constraint $R^{pos}_{NN} \cap R^{sim}_{love,n} \cap R^{rtm}_{01}$. Note that the size n of the semantic constraint is a global parameter, that can be set in the Perec's "Option" menu.
If `<pos>`, `<concept>` or `<rhythm>` are empty, this means that the corresponding constraint is not imposed.
For example `//01` imposes only the rhythmic constraint R^{rtm}_{01}. If `<rhythm>` is empty the second `/` in the constraint definition can be omitted. If both `<concept>` and `<rhythm>` are empty, both the first and second `/` can be omitted, except in the case where also `<pos>` is empty (i.e. no constraint is imposed). In this case at least one `/` is need to inform Perec that no constraint is imposed at this position.
For example the string `DT NN/love /` imposes R^{pos}_{DT} on the first word, $R^{pos}_{NN} \cap R^{sim}_{love,n}$ on the second word and nothing on the third word.
The rhyme constraints are a special case, as they can occur only once at the end of a sequence. A rhyme constraint is of the form `#<rhyme>` and has to be the last constraint of the string to pass to the constraint terminal. As expected, `#<rhyme>` imposes R^{rhy}_{rhyme}.
For example the string `DT NN/love VBZ #hello` imposes R^{pos}_{DT} on the first word, $R^{pos}_{NN} \cap R^{sim}_{love,n}$ on the second word and $R^{pos}_{VBZ} \cap R^{rhy}_{hello}$ on the third word.
Once the good string is entered in the constraint terminal, by pressing Enter on the keyboard or the button 💬 Perec will translate them in the corresponding sequence of control constraints and use it to generate verses. These verses are then proposed to the user as in the case of the statement and continuation buttons. If the string passed in the constrain terminal is not well formed and Perec cannot understand it, an error message is showed to the user. At this point it is important to stress that, using the constraint termi-



Type: Action

Min Lenght: 3

Max Lenght: 10

Min Frequency: 4

Template	Fr...	Le...	Type	First Tag
PRP VBD DT JJ NN	11	5	A	PRP
JJ NNS VBN NNS	9	4	A	JJ
PRP VBD PRP IN DT NN	8	6	A	PRP
PRP MD RB VB	7	4	A	PRP
PRP VBP VBG IN DT NN	7	6	A	PRP
PRP VBD IN DT NN	7	5	A	PRP
VBP RB VB PRP VB	6	5	A	VBP
VBP RB VB VB NN VBP RB VB VB VBN	6	10	A	VBP
PRP VBD PRP$ NN	6	4	A	PRP
PRP MD VB PRP IN DT NN	6	7	A	PRP
RB PRP VBD IN DT NN	6	6	A	RB
IN PRP$ NN PRP VBP VBN VBN	5	7	A	IN
UH VBP PRP MD VB PRP DT NN	5	8	A	UH
PRP VBP DT JJ NN	5	5	A	PRP

Figure 6.4: The template selector allows the user to access the template library and select one template according to various properties.

nal, it is possible to create a sequence of constraints for which the induced POS template does not belong to the template library of the current corpus. This will not raise an error and Perec will try to generate verses that satisfy the imposed constraints, if such verses exist. If the solution space is empty, Perec will inform the user that the imposed constraint are to strict.

Since sometimes can be hard to enter a well formed string to the constraint terminal, or simply the user want to exploit the template library of the corpus, two tools are provided: the *template selector* and the *graphical constraint manager*.

The template selector (showed in figure 6.4) can always be called by pressing on the button . The template selector allows the user to access the template library and select one template according to various properties such as: the template's type (action if the template contains a verb, description otherwise), the length or the frequency, i.e. how many time it appears in the

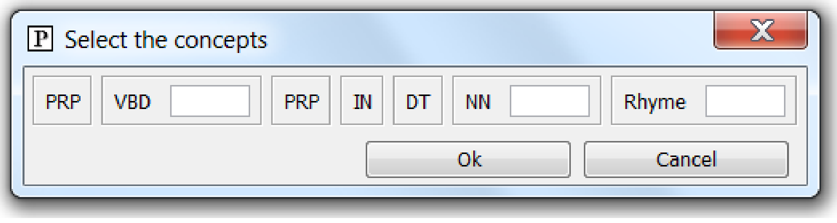


Figure 6.5: The graphical constraint manager provides a graphical, more intuitive way to modify the string that will be passed to Perec via the constraint terminal.

corpus. Once a template is selected, the constrain terminal is filled with the corresponding string. Therefore the template selector is a very useful tool to send a well formed string to the constraint terminal.

The second tool is the graphical constraint manager. It can be called by pressing the button . This button is enabled only if the string in the constraint terminal is not empty. In fact the graphical constraint manager provides a graphical, more intuitive way to modify the string that will be passed to Perec via the constraint terminal. When the graphical constraint manager is called, it parses the string in the constraint terminal and, if the string is well formed, translates it in a more graphical form. One example of the action of the graphical constraint manager is showed in figure 6.5. This graphical form can be edited by the user to modify the imposed constraints. Each modification in the graphical constraint manager will be translate in a modification of the string in the constraint terminal. At the moment the graphical constraint manager does not support rhythmic templates. To easily generate verses that satisfy user defined rhythmic templates, Perec exploits an other plug-in, the *rhythm manager*, described below.

Therefore It is useful to exploit together the template selector and the graphical constraint manager to define constraint to pass to Perec via the constraint terminal. First, a POS template is selected using the template selector, then this template is enriched with other constraints using the graphical constraint manager.

6.4 Other plug-ins

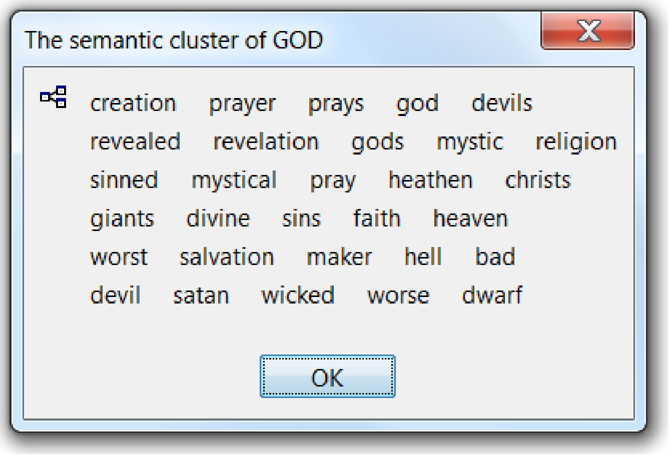
Using the statement and continuation buttons and the constraint terminal is possible to define which constraints to impose on the suggestion generated by Perec at many level, from a very local to a very global level, but sometimes is still difficult to decide what are the best constraint to impose to have the best suggestions from Perec. In fact if the constraints are too strict, no verse that satisfies them exists. On the other hand if the constraints are too permissive, too many verses are generated and the interesting ones are drowned among the uninteresting ones. Moreover, depending on the corpus used, the same sequence of control constraints can lead to completely different results. Therefore Perec comes with some plug-ins designed to help the user to select the good combination of control constraints.

6.4.1 Semantic cluster viewer

The first of such plug-ins is the *semantic cluster viewer*. The semantic viewer is a tool to inspect semantic constraints. It is called by pressing on the button on the toolbar. When the semantic cluster viewer is called, it ask the user what is the concept to inspect. Once a concept c is given, all the words in $R^{sim}_{c,n}$ are displayed on the screen. Figure 6.6 shows an example of semantic cluster viewer. The corpus used is the Dylan corpus, the selected concept is “God” and the size n is set to 30.

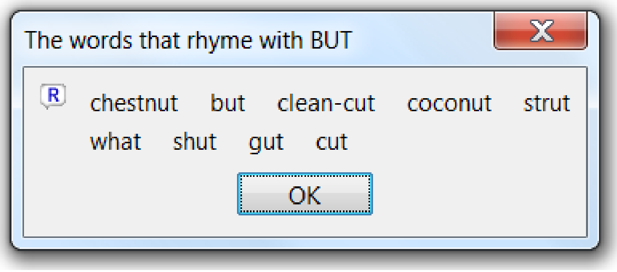
6.4.2 Rhyme viewer

The *rhyme viewer* is the equivalent to the semantic viewer in the case of rhyme constraints. It is called by pressing on the button R on the toolbar. When the rhyme viewer is called, it ask the user what is the rhyme to inspect. Once a rhyme r is given, all the words in R^{rhy}_{r} are displayed on the screen. Figure 6.7 shows an example of rhyme viewer. The corpus used is the Dylan corpus, the selected rhyme is “but”.


The semantic cluster of GOD

creation prayer prays god devils
revealed revelation gods mystic religion
sinned mystical pray heathen christs
giants divine sins faith heaven
worst salvation maker hell bad
devil satan wicked worse dwarf

Figure 6.6: The semantic cluster viewer. It shows the words in $R^{sim}_{god,30}$ in the Dylan corpus, i.e. the 30 words in the Dylan corpus most semantically related to the concept “God”.


The words that rhyme with BUT

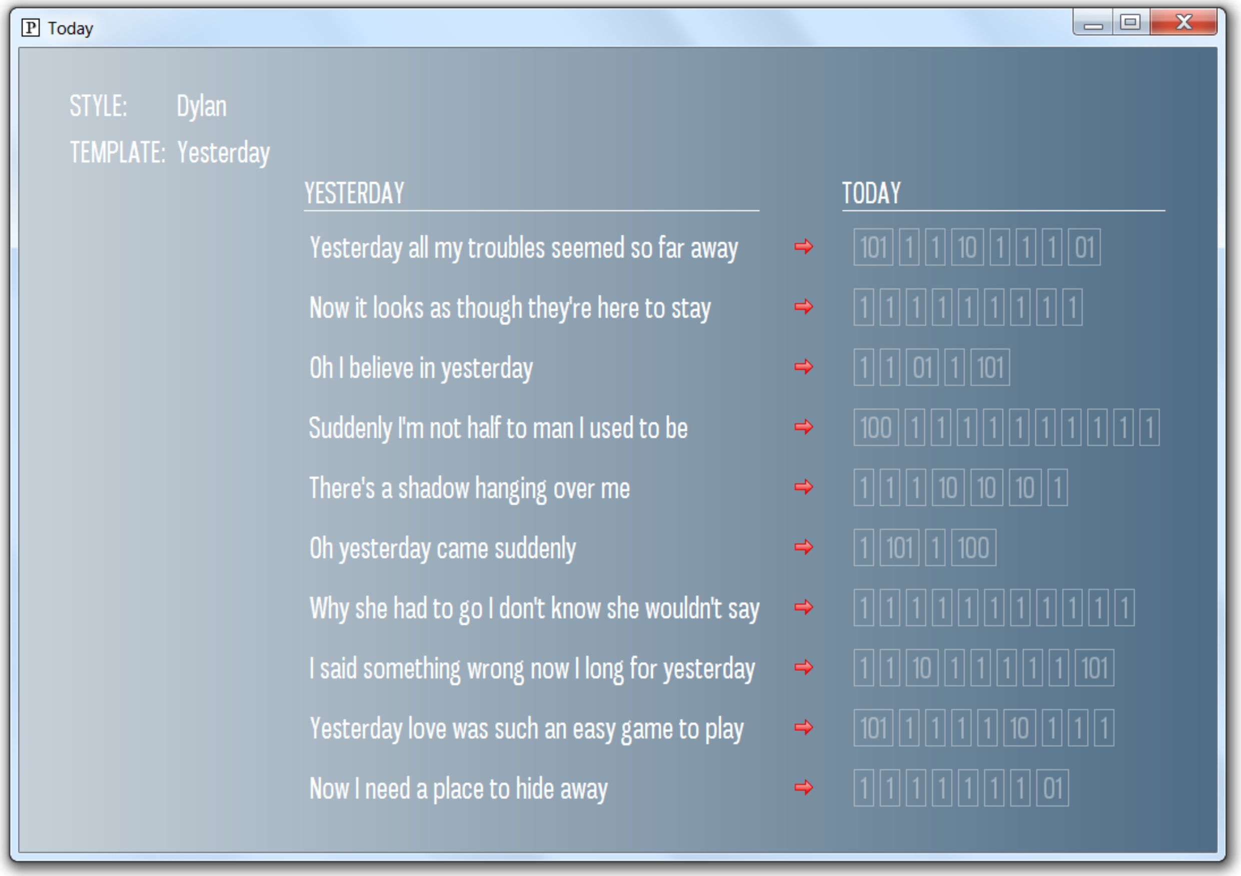
chestnut but clean-cut coconut strut
what shut gut cut

Figure 6.7: The rhyme viewer. It shows the words in R^{rhy}_{but} in the Dylan corpus, i.e. all the words in the Dylan corpus that rhyme with “but”.

6.4.3 Rhythm manager

The rhythm manager is the most extended plug-in, in fact it can ever thought as an editor itself. The rhythm manager is designed to be used when the rhythm is an important feature of the verses to generate. It is called by pressing on the button ⋯ on the toolbar. When the rhythm manager is called, it ask the user for an existing song to be used as a global rhythmic template in the following way: provided the song, the rhythm manager extracts the rhythmic templates of each verse in the input song, and use them as a skeleton for new lyrics to be generated. Figure 6.8 shows the global rhythmic template extracted from the song "Yesterday" by The Beatles.

Then the user can interactively set other control constraints on this skeleton, by clicking on the slot in the various rhythmic templates and adding constraint using the same language used to impose constraints through the constraint terminal. Once all the control constraint are set, it is possible to tell to the rhythm manager to generate verses by clicking on the red arrows. The rhythm manager was used to generate the lyrics of "Today", the song showed in section 4.1.3. It is possible to watch a video showing the whole generation process on the web, at `http://www.csl.sony.fr/MarkovCt/lyrics/`.


Today

STYLE: Dylan

TEMPLATE: Yesterday

YESTERDAY		TODAY
Yesterday all my troubles seemed so far away	⇨	101 1 1 10 1 1 1 01
Now it looks as though they're here to stay	⇨	1 1 1 1 1 1 1 1 1
Oh I believe in yesterday	⇨	1 1 01 1 101
Suddenly I'm not half to man I used to be	⇨	100 1 1 1 1 1 1 1 1 1 1
There's a shadow hanging over me	⇨	1 1 1 10 10 10 1
Oh yesterday came suddenly	⇨	1 101 1 100
Why she had to go I don't know she wouldn't say	⇨	1 1 1 1 1 1 1 1 1 1 1
I said something wrong now I long for yesterday	⇨	1 1 10 1 1 1 1 1 101
Yesterday love was such an easy game to play	⇨	101 1 1 1 1 10 1 1 1
Now I need a place to hide away	⇨	1 1 1 1 1 1 1 01

Figure 6.8: The rhthm manger. In this picture it is ready to used to generate lyrics in the style of Bob Dylan that follows the same structure of the lyrics of the song "Yesterday" by The Beatles.

Chapter 7

Conclusion

This thesis demonstrated that constrained Markov processes can be used to generate texts that imitate a given style while satisfying structural properties. This is achieved by proposing a general solution to the issue of controlling finite-length sequences generated by Markov processes. The solution exploits the fruitful connection between Markov processes and constraint satisfaction, initiated in [33]. The thesis shows that for constraints that remain within the Markov scope, directed arc-consistency enables to compile control constraints in the form of a non-homogeneous Markov model. This model can in turn be used straightforwardly with random walk to generate texts that satisfy the constraints with their original probabilities.
Then the thesis shows how high level properties of a poetic texts, such as syntactic correctness, rhymes and meter can be formulated as unary control constraints. Moreover this same framework enables the verses to be semantically biased towards a given semantic constraint. Once these control constraints are defined, they are used to generate poetic texts and lyrics. First, the semi-automatic generation of lyrics in the style of the songwriter Bob Dylan that has the same structure as an existing song (Yesterday by The Beatles) is described. Then several examples in the style of more 60 authors are listed, demonstrating how this approach can be used to create the lyrics of a song that is both stylistically coherent while satisfying structural

constraints.
The control constraints are then evaluated empirically. The evaluation is done by asking humans to evaluate texts generated by constrained Markov processes against texts generated by two other approaches: a pure Markov approach and a pure constraint satisfaction approach. This evaluation shows that constrained Markov processes generate better texts in terms of syntactic correctness and semantic relatedness.
Finally an augmented text editor, called Perec is described. Perec exploits constraint Markov process to help the user to write poems and lyrics. In fact Perec can suggest at verses that satisfy control constraint proposed by the user.

This thesis has shown that constrained Markov processes can fruitfully be used to generate lyrics in a given style that also satisfy control constraints imposed by the user. However constrained Markov processes ensure that the generated texts are "in the style of" only from a computational point of view. Therefore, in the future, it will be interesting to study how control constraints impact the perceived style of the generated texts. An other possible future work will concern the impact of Perec on the writing style of the user. Can Perec improve the creativity of the user? Do some styles improve the creativity more than others? And what happens if the style set in Perec is the style of the user itlself? Some experiments should be designed to try to answer all these question.

Bibliography

[1] E.G. Altmann, G. Cristadoro, and M. Degli Esposti. On the origin of long-range correlations in texts. *PNAS*, 2012.

[2] E. Alvarez-Lacalle, B. Dorow B, J.P. Eckmann, and E. Moses. Hierarchical structures induce long-range dynamical correlations in written texts. *Proc. Nat. Acad. Sci. USA*, 103:7956–7961, 2006.

[3] D. Appelt. *Planning English Referring Expression.* Cambridge University Press, New York, 1985.

[4] Shlomo Argamon, Kevin Burns, and Shlomo Dubnov. *The Structure of Style: Algorithmic Approaches to Understanding Manner and Meaning.* Springer, 2010.

[5] R. Barzilay. Probabilistic approaches for modeling text structure and their application to text-to-text generation. In *Empirical methods in natural language generation*, pages 1–12. Springer-Verlag, 2010.

[6] A. Budanitsky and G. Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Comput. Linguist.*, 32(1):13–47, 2006.

[7] C.Basile, D.Benedetto, E. Caglioti, and M. Degli Esposti. An example of mathematical authorship attribution. *JOURNAL OF MATHEMATICAL PHYSICS*, 49:1–20, 2008.

[8] A. Davey. *Discourse production.* Edinburgh University Press, Edinburgh, Scotland, 1979.

[9] S. Davismoon and J. Eccles. Combining musical constraints with Markov transition probabilities to improve the generation of creative musical structures. *EvoApplications*, 2:361–370, 2010.

[10] Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34(1):1–38, 1987.

[11] K. Deemter, M. Theune, and E. Krahmer. Real versus template-based natural language generation: A false opposition? *Computational Linguistics*, 31(1):15–24, March 2005.

[12] S. Dubnov, G. Assayag, O. Lartillot, and G. Bejerano. Using machine-learning methods for musical style modeling. *IEEE Computer*, 10(38), 2003.

[13] A. Eigenfield and P. Pasquier. Realtime generation of harmonic progressions using controlled Markov selection. In *Proc. of 1st Int. Conf. on Computational Creativity*, pages 16–25, Lisbon, Portugal, 2010. ACM Press.

[14] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, 29(1):24–32, 1982.

[15] P. Gervás. Wasp: Evaluation of different strategies for the automatic generation of spanish verse. In *Time for AI and Society*, pages 93–100, 2000.

[16] P. Gervás. An expert system for the composition of formal spanish poetry. *Journal of Knowledge-Based Systems*, 14(3-4):181–188, 2001.

[17] J. A. Goguen and D. Fox Harrell. Style as a choice of blending principles. In *Style and Meaning in Language, Art Music and Design*, pages 49–56. AAAI Press, 2004.

[18] Eli Goldberg, Norbert Driedger, and Richard I. Kittredge. Using natural-language processing to produce weather forecasts. *IEEE Intelligent Systems*, 9(2):45–53, 1994.

[19] N. GoldMan. Conceptual generation. *Conceptual information processing*, 1975.

[20] D. Jurafsky and J. H. Martin. *Speech and Language Processing.* Prentice Hall, 2009.

[21] L. Kolarov and Y.G. Sinai. *Theory of Probability and Random Processes.* Springer, 2007.

[22] R. Kurzweil. Cybernetic poet. http://www.kurzweilcyberart.com, 2001.

[23] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[24] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65–74, 1984.

[25] W. Mann. The anathomy of a systemic choice. *Technical report*, 1982.

[26] W. Mann and S. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.

[27] K. McKeown. *Text Generation.* Cambridge University Press, Cambridge, 1985.

[28] D. Milne and I. H. Witten. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *Proceedings of AAAI 2008*, 2008.

[29] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7(66):95–132, 1974.

[30] Y. Netzer, D. Gabay, Y. Goldberg, and M. Elhadad. Gaiku: Generating haiku with word associations norms. In *Proc. of the Workshop on Computational Approaches to Linguistic Creativity (CALC)*, pages 32–39. ACL Press, 2009.

[31] H. R. Gonçalo Oliveira, F. Amílcar Cardoso, and F. C. Pereira. Tra-la-lyrics: an approach to generate text based on rhythm. In *Proc. of the 4th. International Joint Workshop on Computational Creativity*, 2007.

[32] F. Pachet. The continuator: Musical interaction with style. In *Proceedings of ICMC*, pages 211–218, Goteborg, Sweden, 2002.

[33] F. Pachet, P. Roy, and G. Barbieri. Finite-length Markov processes with constraints. In *Proc. of IJCAI 2011*, pages 635–642, Spain, 2011.

[34] R. Power, D. Scott, and N. Bouayad-Agha. Generating texts with style. In *Proc. of the 4th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing03)*, pages 444–452, Mexico, 2003.

[35] F. Rahman and H. M. Manurung. Multiobjective optimization for meaningful metrical poetry. In *Proc. of ICCC-11*, pages 4–9, Mexico, 2011.

[36] E. Reiter and R. Dale. *Building Natural Language Generation Systems*. Cambridge University Press, 2000.

[37] E. Reiter and S. Williams. Generating texts in different styles. In *The Structure of Style: Algorithmic Approaches to Manner and Meaning*, pages 59–75. Springer-Verlag, 2010.

[38] A. Rudnicky. The Carnegie Mellon pronouncing dictionary, version 0.7a. http://www.speech.cs.cmu.edu/cgi-bin/cmudict, 2010.

[39] B. Settles. Computational creativity tools for songwriters. In *Proc. of the NAACL-HLT Workshop on Computational Approaches to Linguistic Creativity*, pages 49–57. ACL Press, 2011.

[40] H. Somers. Review article: Example-based machine translation. *Machine Translation*, 14:113–157, 1999.

[41] Efstathios Stamatatos. A survey of modern authorship attribution methods. *J. Am. Soc. Inf. Sci. Technol.*, 60(3):538–556, March 2009.

[42] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984.

[43] K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc.of HLT-NAACL 2003*, pages 252–259, 2003.

[44] J. Wagner, J. Foster, and J. van Genabith. Judging grammaticality: Experiments in sentence classification. *CALICO Journal. Special Issue on the 2008 Automatic Analysis of Learner Language*, 2009.