Constraints (2011) 16:148–172
DOI 10.1007/s10601-010-9101-4

Markov constraints: steerable generation of Markov sequences

François Pachet · Pierre Roy

Published online: 18 September 2010

Abstract Markov chains are a well known tool to model temporal properties of many phenomena, from text structure to fluctuations in economics. Because they are easy to generate, Markovian sequences, i.e. temporal sequences having the Markov property, are also used for content generation applications such as text or music generation that imitate a given style. However, Markov sequences are traditionally generated using greedy, left-to-right algorithms. While this approach is computationally cheap, it is fundamentally unsuited for interactive control. This paper addresses the issue of generating *steerable Markovian sequences*. We target interactive applications such as games, in which users want to control, through simple input devices, the way the system generates a Markovian sequence, such as a text, a musical sequence or a drawing. To this aim, we propose to revisit Markov sequence generation as a branch and bound constraint satisfaction problem (CSP). We propose a CSP formulation of the basic Markovian hypothesis as elementary Markov Constraints (EMC). We propose algorithms that achieve domain-consistency for the propagators of EMCs, in an event-based implementation of CSP. We show how EMCs can be combined to estimate the global Markovian probability of a whole sequence, and accommodate for different species of Markov generation such as fixed order, variable-order, or smoothing. Such a formulation, although more costly than traditional greedy generation algorithms, yields the immense advantage of being naturally steerable, since control specifications can be represented by arbitrary additional constraints, without any modification of the generation algorithm. We illustrate our approach on simple yet combinatorial chord sequence and melody generation problems and give some performance results.

F. Pachet (✉) · P. Roy
Sony CSL, 6, rue Amyot, Paris, France
e-mail: pachet@csl.sony.fr

P. Roy
e-mail: roy@csl.sony.fr

Keywords CSP · Markov chains · Sequence constraints · Global constraints · Interactive applications

1 Introduction and motivation

In this section, we describe the context in which the problem of controlling Markovian sequences arises. We then introduce Markovian sequence generation formally. In a second part, we review the state of the art in using CSP for sequence generation.

1.1 Motivation

Markovian sequence generation is a well known modeling tool used in many content generation applications, such as text generation, music composition and interaction and style imitation in general. Since Shannon's seminal works on information theory, the Markovian aspects of musical sequences has long been acknowledged, see e.g. [8]. Many attempts to model musical style have therefore exploited Markov chains in various ways [21]. In particular, *Variable-Length Markov Models* (in short, VMMs) have been used successfully for music analysis and composition [5]. The work presented here draws from the many experiments conducted with this technique for music modeling and interaction.

The Continuator [22] was the first system in this species to learn and react interactively to human music input. Its success was largely due to the capacity of the system to faithfully imitate arbitrary musical styles, at least for relatively short time frames. This capacity is in turn related to the fact that the "Markovian hypothesis" basically holds for most of pitch sequences played by users (from children to professionals) in many styles, notably belonging to tonal music (Classical, Jazz, Pop, etc.).

With the Continuator, a user typically plays a musical phrase using a Midi instrument (e.g. a keyboard). This phrase is then converted into a sequence of symbols, representing a given dimension or *viewpoint* of music, such as its pitch, duration, or velocity. The sequence is then "learnt" by the system by computing a model of the transition probabilities between successive symbols. When the phrase is finished (typically after a certain temporal threshold has passed), the system generates a new phrase using the Markov model built so far. The user can then play another phrase, or interrupt the phrase being played, depending on the chosen interaction mode. Each time a new phrase is played, the Markov model is updated. It was shown that such an incremental learning creates engaging dialogs with users, both with professional musicians and children [1]. Other systems such as Omax [11] followed the same principle with similar results.

In such an interactive context, the *control problem* arises naturally. By construction, generated phrases are Markovian (in a sense that we define below precisely) and therefore perceived as stylistically consistent. But the user is not able to specify additional properties he/she wishes these sequences to satisfy. These control properties are not artificial, and can be of many kinds. For instance, in a musical context, the user may want the sequence to be globally ascending, pitch-wise, or to satisfy an arbitrary pitch *contour*. This control property can be a consequence of a particular gesture, detected, e.g. by input sensors. Other properties can be related to musical knowledge. For instance, it is often interesting in music improvisation to generate a sequence that ends on a specific note. This is particularly visible when observing, e.g., guitar virtuoso improvisations in the style of Hard-Rock, which often consist of a frantic sequence of notes ending gracefully on a note of the triad (e.g. the tonic of the underlying chord). The control problem arises for other dimensions of music, such as harmony. In the context of (tonal) chord generation, e.g. [14], it is natural to add additional constraints to sequences generated from a Markov model. For instance, one can look for chord sequences whose last chord resolves on to the first, in order to convey a sense of tonality, or avoid too long repetitions of the same chord, or impose certain chords to be at certain positions in the sequence, to force specific modulations to occur.

The main problem we face here is that these control properties establish explicit relationships between items in a sequence that cannot be represented by a local Markov hypothesis (as described below). A claim of this paper is that the very issue of controlling a Markovian sequence generator is necessarily combinatorial, and cannot be satisfactorily addressed by purely greedy algorithms, or by tweaking Markov models.

Another important point in our context is that real-time performance imposes limitations on the length of the sequences to generate. In practice, we consider a sequence generation scheme in which sequences are built chunk by chunk, and with user control holding only within a chunk. We therefore look for methods that are as expressive as possible constraint-wise, while possibly not scaling well for long sequences (e.g. fully-fledged musical pieces).

In the next section we describe formally Markov generation using the traditional approaches, stressing on the difficulty to satisfy non local properties of the sequence.

1.2 Various species of Markov sequence generation

Markov sequences represent stochastic processes having the "Markov property" [6]. This property says that the future state of the sequence depends only on the last state, i.e.:

$$P(s_i|s_1, \cdots, s_{i-1}) = P(s_i|s_{i-1})$$

Markov chains have been extensively studied in probabilities, notably to study properties of infinites sequences (such as convergence, existence of cycles, etc.). In practice, order-d Markov chains are often used as a generalization, by considering d states in the past instead of one. An order-d Markov chain (also known as *d-gram*) models the future of a partial sequence from its immediate past of length d, approximating:

$$P(s_i|s_1, \cdots, s_{i-1}) = P(s_i|s_{i-d}, \cdots, s_{i-1})$$

In theory, order d Markov chains can be translated into larger chains of order 1, by considering a product state space. In practice however, considering orders larger than 1 yields a better compromise, notably because there are many ways to represent efficiently the set of continuations for all possible prefixes of a given sequences set (such as prefix trees or Factor Oracles [2]).

In the case of a Markov chain of fixed order 1, one can define the log-probability of a finite sequence of length N as the product of the logarithm of each probability:

$$(Log - probability\ of\ a\ finite\, sequence) \qquad Log\,(P\,(S)) = \sum\nolimits_{k=1}^{N} Log\,(P\,(s_{k+1}|s_k))$$

This definition extends trivially to orders larger than 1.

A more expressive class of Markov chains is variable-length Markov models (VMM, [31]). In this model, chains of varying orders are considered, resulting in the capacity for the model to capture statistical correlations of different length scales in a single probabilistic model [6]. For many applications it was shown that VMMs offer a better compromise than Hidden Markov Models (HMMs) in terms of simplicity of use [25, 31].

For the applications we target, the Markov model is estimated from a *training set*, i.e. a set of sequences deemed representative of the corpus of study. Transition probabilities are estimated by counting the number of occurrences of the various states, considering all possible subsequences of length $d - 1$ in the training set. Initial probabilities are usually estimated by counting the number of occurrences of each item. Our goal is to generate a sequence of finite length by exploiting a Markov model obtained after the analysis of a training set.

In practice, Markovian generation, for all species, is based on a "random walk approach" [9] in which sequences are generated step-by step, by the following greedy process. At step i, the generated sequence s is noted $s_1, s_2, \cdots, s_i$. An appropriate structure (such as a transition matrix, a prefix tree or a graph representing the possible continuations for all possible subsequences of the training set) yields the set of all possible continuations for the subsequence (prefix) of s considered. This prefix can be a subsequence of fixed size (in the case of fixed-order Markov generation), or variable size in the case of VMMs. In the latter case the prefix considered can be for instance the longest one for which there exists at least one possible continuation, starting from the end, i.e. $\mathrm{s_{i-p}}, \mathrm{s_{i-p+1}}, \ldots, \mathrm{s_i}$. This is the strategy adopted in the Continuator [22]. Other strategies can be used, such as the one described in [25] who propose a combination of the probabilities of continuations of various sizes, a technique referred to as "smoothing".

Once a prefix *Pref* is determined, a continuation Y is chosen by a random draw in this continuation set, weighted according to the probabilities of each continuation for this prefix:

$$\text{(E)} \quad P\,(Y) = P\,(Y|Pref)$$

This continuation is then appended to s, and the process is iterated with the expanded s as a new input sequence. This algorithm is simple to implement, hence its success for interactive applications as described above.

It is important to note that random walk approaches favor, by definition, the most probable local continuations, whatever the species of Markov chain considered. As we will see, adding control constraints necessitates the exploration of sequences built with possibly very small continuation probabilities. Such sequences are very unlikely to be generated by a greedy algorithm.

Whatever Markov species is considered (order 1, d or variable) the random walk approach to Markov generation has two related drawbacks. The first one is theoretical, and applies mainly to variable-order generation: sequences generated using random walk variable-order are not the most probable, as emphasized in

[9]. This is due to the fact that only local decisions are made at each step. In our context, this is not necessarily a problem since we do not want to generate complete sequences, but rather generate them chunk by chunk, as described below. A more serious limitation of this approach, addressed in this paper, is that sequences generated by random walk cannot be controlled. By "control" we mean additional, context-dependent properties that the user wants the generated sequence to satisfy. Because random walk generates sequences step by step with no backtracking, it is impossible to specify control properties that would hold beyond the next item to generate. More generally as we will see, properties holding on arbitrary subsets of items cannot be taken into account by random walk approaches.

1.3 Controlling Markov chain generation

Controlling a Markov generation algorithm creates indeed a paradox. Imposing a constraint holding on one or several elements of the sequence to generate implies that only a subset of the modeled corpus should be considered: the set of sequences satisfying the constraints. However, this subset is not necessarily Markovian. For instance, imposing that the first and last elements are equal obviously violates the Markovian hypothesis of limited temporal dependency (as soon as the length S of the sequence to generate is greater than d, the maximum order considered). Even simple *anchor* constraints, i.e. unary constraints holding on one particular element of the sequence will in general produce subsets that are not Markovian. So it is meaningless to consider the "Markov model" of this subset in general. The practical goal we have is therefore to search for finite-length sequences which (1) satisfy the control constraints while (2) being optimally Markovian, in the sense of the "original" model.

An attempt to control VMM sequences for a specific case was proposed in [22]. This method consists in biasing the Markovian probabilities used during the generation phase by an external function, typically produced by the user. In this mode the user plays an (arbitrary) melody and the system generates a note-to-note real-time harmonization of the melody, i.e. plays a chord for each note played. For each note, a chord is chosen using the random walk approach, in order to build a Markovian sequence from a previously learnt training set of chord sequences. The probability is biased to take into account the current note being played, so as to favor chords that fit harmonically with this note (for instance, chords containing this pitch). As a result, the system generates real-time harmonisations having a Markovian quality while fitting with the played melody. However, this approach cannot be used to satisfy constraints holding on several items at the same time, or holding on an item at a fixed, future position (e.g. the last note), as it does not involve any backtracking.

More recently [11] proposed to graft a reinforcement learning stage on top of a VMM generation algorithm, to produce music material exhibiting long-term structure, in an attempt at better modeling musical composition and improvisation behavior. In a related domain, [29] use constraint programming to solve guitar fingering problems, i.e. reconstruct the gesture from a musical score, but they consider only local constraints (holding on chords).

A more general framework for representing complex signals are Hidden Markov Models (HMM, [28]). In HMM, hidden (not observable) states are added, as a way to better represent the context. Observable states can be considered as specific control properties. For example HMMs were used to generate harmonizations of given melodies (e.g. [15]). In the graphical domain, HMMs are used to improve line sketchings in [33]. A rough sketch is first drawn by the user (e.g. a cloud). The sketch is then used to estimate the best possible set of "refined sketches", previously learnt and represented as a HMM (e.g. a "cloud HMM"). A new refined curve is then generated from the rough sketch and the selected HMM using a Viterbi algorithm.

It is important to note that the HMM approach works thanks to the Bellman principle. The reason why Bellman holds is that observable states can be used as an *objective function*, i.e. an accumulative sum of monotonic, non-decreasing cost functions. As such, this approach cannot be used to model general control constraints, notably constraints that cannot be estimated *a priori*. Obviously, constraints holding on more than one item cannot be satisfied with a greedy algorithm. However, in the context of Markovian sequence generation, even simple constraints such as anchor constraints (imposing a fixed value at a specific position) may have implications on the whole sequence, i.e. influence more than d neighbors of the anchor. Therefore the Bellman principle behind Viterbi does not hold, and some form of combinatorial search is required to generate a Markovian sequence that satisfies the constraints.

This paper addresses exactly this problem: how to generate a sequence that is Markovian while satisfying additional arbitrary constraints. The solution we propose is to look at Markovian sequence generation from a constraint programming perspective. In our approach, arbitrary control constraints are easy to introduce, thanks to the huge existing constraint library, e.g. the *AllDiff* constraint [30], *sequencing* constraints [35], and more generally global constraints [7]. Before describing our solution, we review below the existing CSP approaches in sequence generation.

1.4 Sequencing constraints

Constraint programming was extensively used for melody harmonization problems, see [4]. Constraint programming was so far less considered for interactive, real time applications. Anders and Miranda [3] proposed a general framework to use CSP for real time music composition, but addressed essentially the time-out problem to use CSP in a traditional manner. Sequence generation using CSP was used in [23] to build interactively music playlists. A Markovian property holding on consecutive titles in a playlist was considered but only addressed binary contiguity constraints, i.e. order-1 Markov sequences.

On the other hand, sequence generation has been studied in the context of global constraints for operation research applications such as scheduling or rostering. A famous constraint dealing with sequence generation is the *sequencing* constraint [7, 36]. This constraint defines sliding cardinality relations holding on the possible values of variables. As such it does not address our problem.

The *regular* constraint [27] and its various extensions (*cost*, *cost-regular*) are constraints holding on a fixed-length sequence of variables, requiring that the corresponding sequence of values belong to a regular language, itself defined by a deterministic finite state automaton (DFA). Regular constraints could be used, as an ingredient in our solution, but we propose a simpler and more efficient approach, as discussed in Section 3.4.

Another related constraint is the constraint defined in extension, i.e. by the set of admissible tuples, such as the IlcTableConstraint [16]. As we will see below, although we define our Markov constraints in extension, the need to cope with variable-orders imposes additional requirements for the filtering algorithms.

2 Examples

In this section we introduce two typical examples of Markovian music generation: chord sequences and melodies. We describe shortly the problems and corpuses used for training, and show some typical examples of control constraints and solutions obtained by our method.

2.1 Chord sequence generation

The generation of harmonic structures has long been recognized as style-dependent. The specific case of Blues chord sequences has received a lot of attention in music modeling, since the works of [34], who emphasized the grammatical structure of 12 bar blues. Grammar-based approaches consist in representing explicitly the corpus of all sequences as a set of rules, justified by harmonic knowledge. However, this approach does not allow users to control the generated sequences easily. Moreover, grammar-based approaches do not naturally cope with transition probabilities, and all sequences are treated as equally probable.

We propose here to generate 12-bar Blues sequences from a Markovian perspective, to illustrate the new possibilities offered by controlled Markovian generation. We consider a corpus of Blues chord sequences taken from Charlie Parker's Omnibook [24]. Each chord is represented by a symbol consisting of (1) the pitch class (12 possibilities) and (2) the chord type. We restrict this study to three chord types: Seventh (noted 7), minor (noted *min*) and half diminished (noted *h*7). 22 Blues are taken as representative of Charlie Parker's "Blues" style. A typical example of a Blues chord sequence is "*Blues for Alice*":

$$F7|Eh7/A7|Dmin/G7|Cmin/F7|Bb7|Bb\,min/Eb7|Amin|Ab\,min/Db7|Gmin|\\ C7|F7|Gmin/C7$$

where vertical lines represent bars, and slashes represent half bars. In this paper, we systematically consider Blues as sequences of 24 chords with two chords per bar, by repeating chords when needed. "Blues for Alice" is thus eventually represented as:

F7 F7 Eh7 A7 Dmin G7 Cmin F7 Bb7 Bb7 Bbmin Eb7 Amin Amin Abmin Db7 Gmin Gmin C7 C7 F7 F7 Gmin C7

Another example taken from our corpus is *BackHomeBlues*, represented as:

C7 C7 C7 C7 C7 C7 C7 C7 F7 F7 F7 F7 C7 C7 Emin A7 Dmin Dmin G7 G7 C7 C7 Dmin G7

The full corpus is given in [19]. In this example, all sequences have been transposed into C. The approach we follow here is to train a Markov model on this Blues sequence corpus, and generate new sequences from this model. Control constraints are then added in order to bias the generation to particular "species" of Blues.

2.1.1 Blues constraints

As an example, the following sequence could be generated from a Markov model of fixed-order 1 using a traditional greedy algorithm (Table 1).

This sequence is clearly not a "Blues" in the traditional sense. For instance, it does not end with a chord that resolves on to the first (like in all examples of the corpus). This is normal, since such a property is not encoded in the Markov model. The structure of Blues sequences can, however, be defined easily by the following set of constraints:

(Blues Constraints):

- Start by a given key (say, "C7").
- End by a chord that resolves on to the first (say, "G7").
- Play the fourth of the initial key at the beginning of the second "line", i.e. at position 9 (here, "F7").

With these constraints, a generated sequence could be the following (Table 2).

2.1.2 Generating exotic blues

More complex control constraints can be added to generate yet other Blues variations. For instance, one could generate a Blues in C in which there is exactly one occurrence of an F#7 chord. This constraint is interesting because F#7 is relatively "far away" from the tonality of C so such a sequence explores the limits of "bluesness".

However, it is impossible to obtain this sequence with the corpus as defined above, because F#7 does not appear in any of the training Blues transposed in C. To increase the possibilities of generation, we translate the corpus in all keys. As a result, each chord has 36 possible values (12 pitch classes and three chord types).

With this transposed corpus, the most probable sequence, with fixed-order 1 given the Blues constraints is the following (Table 3).

With the additional cardinality constraint (exactly one "F#7"), the most probable solution is shown in Table 4. This sequence is 10^6 less probable than the most probable Blues, which shows that satisfying such a constraint would not be reachable by a greedy approach.

Another example is a Blues in which all chords are different. The use of the *AllDif* constraint in music generation has been promoted by the developments of "serial" music. This type of music consists in escaping tonality by treating equally all 12 pitch classes, and has been illustrated by composers like Schönberg, Berg, Webern and later Boulez. For this reason, we call this sequence the "Boulez Blues". This Blues sequence blends in an interesting way two contradictory constraints: the Markovian probabilities which tend to generate a sequence that imitate Charlie Parker's deeply tonal Blues style, and the *AllDiff* constraint that tend to dissipate the sense of tonality. Similarly, its log-probability (−46.74), show that this sequence is particularly difficult to reach. Such a sequence has never, to our knowledge, been exhibited, because it requires the exploration of a huge search space (Table 5).

These examples illustrate the power of controlled Markovian generation for generating harmonic progressions. The same ideas can be used to generate melodies as shown below.

Table 1 A 24 chord sequence generated from a Markov model of fixed order 1, from the Omnibook corpus

(Sequence starting by "C7", no control constraints)			
C7 (0.4)/C7 (0.65)	C7 (0.65)/C7 (0.65)	C7 (0.65)/C7 (0.65)	C7 (0.65)/A7 (0.05)
Dmin (0.73)/G7 (0.62)	G7 (0.5)/G7 (0.5)	G7 (0.5)/C7 (0.46)	C7 (0.65)/F7 (0.13)
C7 (0.26)/Dmin (0.08)	G7 (0.62)/G7 (0.5)	G7 (0.5)/C7 (0.46)	C7 (0.65)/F7 (0.13)

The probabilities of each chord (given the preceding one) are indicated. The log-probability of the sequence (as defined in Section 3.2) is −21.74. It can be observed, however, that this sequence does not satisfy the properties of a Blues. For instance it does not end by a G7

Table 2 A 24 chord sequence generated from the Omnibook corpus, and satisfying the additional "Blues" constraints

(Blues in "C7")			
C7 (0.4)/C7 (0.65)	Bh7 (0.01)/E7 (1)	Amin (1)/D7 (1)	Gmin (0.6)/C7 (0.9)
F7 (0.13)/F7 (0.69)	F7 (0.69)/F7 (0.69)	F7 (0.69)/F7 (0.69)	F7 (0.69)/Fmin (0.06)
Fmin (0.44)/ Emin (0.33)	Emin (0.29)/ Ebmin (0.18)	Ebmin (0.43)/ Dmin (0.43)	Dmin (0.36)/ G7 (0.62)

The log-probability of the sequence is −21.45. The beginning is similar to "Blues for Alice", but the ending is original

Table 3 The most probable "Blues" sequence with a transposed corpus has a log-probability of −20.55

(Optimal Blues in "C7")			
C7 (0.06)/C7 (0.6)	C7 (0.6)/C7 (0.6)	C7 (0.6)/C7 (0.6)	C7 (0.6)/C7 (0.6)
F7 (0.13)/C7 (0.07)	C7 (0.6)/C7 (0.6)	C7 (0.6)/C7 (0.6)	C7 (0.6)/C7 (0.6)
C7 (0.6)/C7 (0.6)	C7 (0.6)/C7 (0.6)	C7 (0.6)/C7 (0.6)	C7 (0.6)/G7 (0.07)

Table 4 A Blues sequence with the additional constraint that it should contain exactly one F# 7 chord

C7 (0.06)	C (0.6)	C7 (0.6)	C7 (0.6)	C7 (0.6)	C7 (0.6)	C7 (0.6)	C7 (0.6)
F7 (0.13)	Bbmin (0.06)	Eb7 (0.57)	Eb7 (0.6)	Eb7 (0.6)	Eb7 (0.6)	Abmin (0.06)	Db7 (0.57)
F#7 (0.13)	Cmin (0.01)	Cmin (0.33)	Cmin (0.33)	Cmin (0.33)	Cmin (0.33)	Cmin (0.33)	G7 (0.01)

The log-probability of the sequence is −34.5, so 10^6 less probable than the most probable Blues

Table 5 The Boulez Blues: a Blues sequence with all different chords, and, here, optimal Markovian probability (fixed order 1)

C7 (0.06)	Fmin (0.06)	Bb7 (0.57)	Ebmin (0.06)	Ab7 (0.57)	Db7 (0.13)	Dbmin (0.01)	Cmin (0.08)
F7 (0.57)	Bbmin (0.06)	Eb7 (0.57)	Abmin (0.06)	Gmin (0.08)	Gbmin (0.08)	B7 (0.57)	Gb7 (0.07)
Bmin (0.06)	E7 (0.57)	Amin (0.06)	D7 (0.57)	Emin (0.04)	A7 (0.57)	Dmin (0.06)	G7 (0.57)

The ending (last row) is interestingly very tonal. The log-probability of the sequence is −46.74, which is 10^{11} less likely than the most probable Blues. There is no solution with fixed-order greater than 1

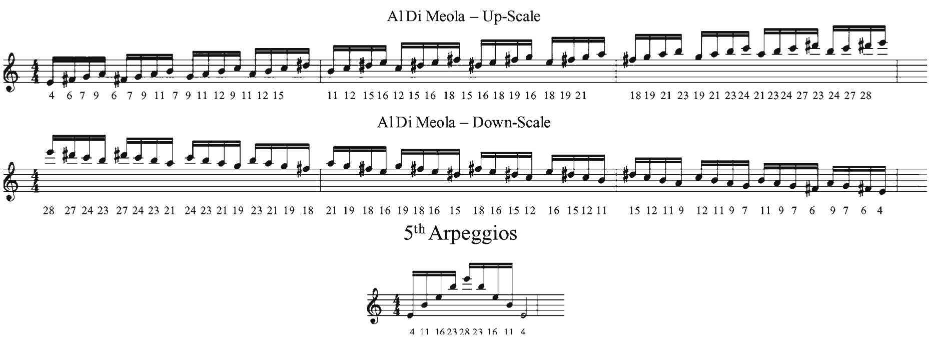


Fig. 1 The three melodies of our training database: *ascending scale*, *descending scale*, and *arpeggio*, all in E minor. The pitch of each note is represented as an integer, with the convention that $C_4 = 0$, $C\#_4 = Db_4 = 1, \ldots, G_4 = 7$, etc. Enharmonically equivalent notes are not distinguished. Durations are ignored in the problem presented here, only pitches are considered. Although these examples are in the E minor scale we notate them in the key of C with explicit accidentals

2.2 Melody generation

In this section, we illustrate controlled Markovian generation for melodic improvisation. In this example, only specific information about melodies is represented. The pitch of each note is represented as an integer, with the convention that $C_4 = 0$, $C\#_4 = Db_4 = 1, \ldots, G_4 = 7$, etc. Enharmonically equivalent notes are not distinguished. Note durations are ignored. In summary, melodic information is represented by integer sequences (pitch sequences). Note that other information could be represented and combined. This aspect, known as the viewpoint problem, is discussed in Section 3.6.2.

The problem we address is to generate a 16 note sequence representing one measure of "improvisation" in the style of the famous virtuoso guitarist Al Di Meola. His style is characterized by the use of specific scales, and transcriptions of his improvisations can be found in many scores (e.g. [13]). In our example, we consider a typical "staircase" Al Di Meola scale consisting of the three following melodies (shown in Fig. 1 using conventional music notation), for a total of 105 notes (15 different notes).

- Melody 1 is an ascending "staircase" scale in the key of E minor and is represented by the following sequence: 4, 6, 7, 9, 6, 7, 9, 11, 7, 9, 11, 12, etc.
- Melody 2 is the descending scale and is represented by the following sequence: 28, 27, 24, 23, 27, 24, 23, 21, etc.

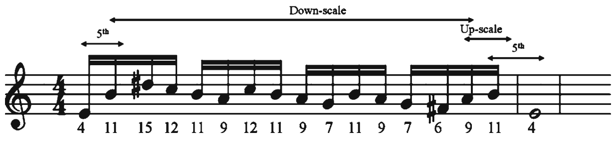


Fig. 2 A 17 note melody with a variable-order Markov (max-order 4), and additional constraints (starts and ends by E). A longer prefix was chosen unintentionally by the system

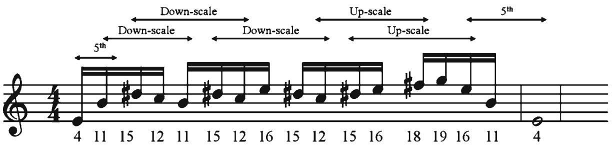


Fig. 3 A 17 note melody with a variable-order Markov (max order 4), and order 5 forbidden, and additional constraints (starts and ends by E). Consequently the melody does not contain any prefix longer than 4

- Melody 3 is a short scale consisting of 5th intervals, still in E minor, and is represented by the following sequence: 4, 11, 16, 23, 28, 23, 16, 11, 4. This melody is added to the corpus to allow intervallic variety.

We now want to generate new melodies in this style. Like in the chord sequence example above, additional constraints are motivated by the musical context. As an example, we consider a simple constraint that the melody should start and end by the same note (E). We give below several examples of such sequences, generated by variable-length Markov models using our approach (described in the next sections) (starts and ends by E, variable-order 4) (Fig. 2).

This example shows the varying size of prefixes used for the generation. Note that a maximum order d (here, 4) corresponds to the size of the learnt Markov model, but not necessarily to the maximum size of prefixes used in the generation, as prefixes of larger size can be unintentionally present. This can lead to the presence of long replicas of the training set, a feature that may be unwanted in some cases. We show here another example where we have forbidden the use of prefixes of size larger than 4, a possibility offered by our approach, as explained in Section 3.2: (starts and ends by E, Variable-order 4, with order 5 disallowed) (Fig. 3).

3 A CSP formulation of Markov sequence generation

A naïve solution to find Markovian sequences satisfying arbitrary control properties is to use generate-and-test. In this approach we define control properties as cost functions. We then generate many sequences using the random walk algorithm, and select the one that minimizes this cost. Obviously, this method is costly with no guarantee to find good, let alone optimal solutions.

The solution we propose consists in doing exactly the opposite. We explore the set of sequences that satisfy exactly the control constraints, and we define the Markovian property as a cost function to optimize. More precisely we explore the space of all sequences of length N belonging to $\sum^N$ where $\sum$ is the alphabet, and N is fixed and is the maximum length of the sequences to generate. Optimization is performed by a Branch and Bound resolution strategy that optimizes the cost function on the sequence to generate. The architecture we propose below can be seen as a way to estimate the cost function efficiently using a CSP approach.

In the next section we define more precisely how to define the Markovian property as a cost function, depending on the species of Markov chain considered.

3.1 Continuations as Elementary Markov Constraints (EMCs)

We represent the notion of Markovian cost for a sequence as a constrained variable to optimize. We can observe that Markov sequences are built in an incremental manner, by considering the continuations of subsequences of a fixed or bounded size.

To represent this generation scheme as a constraint problem, we introduce *Elementary Markov Constraints* (EMCs). An EMC represents a continuation for a given context, together with its probability. More precisely, an EMC of order d represents a sequence of length $d + 1$ that would be produced by a generation algorithm of order d, i.e. a context (d contiguous variables in the sequence), and its continuation (the $(d + 1)^{\text{th}}$ variable in the sequence). Each EMC maintains another variable called *prob*, which represents the estimated probability of the continuation given the context. So each EMC is defined as a triplet:

$$EMC = \{context,\ continuation,\ prob\}$$

The constraint propagators we describe below ensure that $P\,(continuation|context) = prob$, where *prob* represents the probability of the continuation given the context, as estimated from the training set. Once EMCs are introduced, various global cost functions can be defined depending on the applications, and optimized. Additional constraints can then be posted on arbitrary items of the sequence.

EMCs can be combined in various ways, depending on the type of Markovian generator we wish to simulate. We propose below examples with fixed- and variable-order and different generation strategies.

3.2 The Markovian property as a cost function

We define the cost of a sequence s as the sum of Markovian cost for each s_i making up the sequence:

$$cost(s) = \sum\nolimits_{s_i \in s} cost(s_i).$$

By definition, random walk approaches consider individual $cost(s_i)$ at each step, but not $cost(s)$. The approach we propose optimizes $cost(s)$ by searching the sequences that satisfy the control constraints in $\sum^N$. The important issue is how to define $cost(s_i)$. We propose below 4 cost functions, representing the most used strategies in Markov chain generation.

3.2.1 Fixed-order

Given a sequence $s = s_1, s_2, \cdots, s_N$, and a fixed order d, the fixed-order Markovian cost of s is the product of the probability of each item in the sequence. Technically it is more convenient to consider the log-probability to convert multiplications into sums:

$$Cost_{fixed}\,(s_i) = \log\left(P_{fixed}\,(s_i)\right) = \log\left(p\,(s_i|s_{i-d}, \ldots, s_{i-1})\right)$$

When there is no continuation at fixed order d, the corresponding EMC is considered as violated.

In the context of variable-order Markov generation, contexts of varying lengths are considered. There are several ways to define a variable-order cost function, depending on the strategy used.

3.2.2 Smoothing

The *smoothing* technique was introduced originally to cope with the cases when some items in the sequence have no context of order *d*. The probability of those items is 0, which is known as the zero-frequency problem [37]. To solve the zero-frequency problem, the probability of a given item is computed by combining the probability of the item with respect to contexts of different orders. There is no general rule to define the smoothing function, and different functions were introduced [25]. In our examples, we use the following smoothing function, which simply averages the probability of an item over all the orders, from 1 to *d*:

$$Cost_{smooth}(s_i) = \log\left(\frac{\sum_{l=1}^{d} p(s_i|s_{i-l}, \ldots, s_{i-1})}{d}\right)$$

3.2.3 Max-order

Another technique referred to here as “Max-order” consists in choosing always the maximum order possible. This corresponds to the strategy used for the Continuator.

$$Cost_{\max}(s_i) = \log\left(p\left(s_i|s_{i-l\max}, \ldots, s_{i-1}\right)\right)$$

where *lmax* is the maximum order for which $P(s_i|s_{i-l\max}, \ldots, s_{i-1}) \neq 0$

3.2.4 Algebraic

Another interesting cost function consists in considering only the lengths of the prefixes, in a purely algebraic view of Markov chain generation. Such a cost favours the longest but not necessarily the most probable prefixes. This cost function can be defined as follows. The use of square is arbitrary, and favors longer continuations, other polynomials could be used instead:

$$Cost_{alg}(s_i) = lmax^2$$

In the next sections we define more precisely how these cost functions can be represented as constraints holding on the problem variables, and on variables specific to each EMC.

3.3 Problem statement

The generation of sequences is represented as a constraint optimization problem as follows. Let S be a set of training sequences over Σ. We wish to generate sequences of length N that optimize a Markovian cost function while satisfying the control constraints.

Let d be the maximum order of the Markovian constraints. Items of the sequence are represented by finite domain *item variables* $v_1, \ldots, v_n$ whose domain is Σ. Each item variable is constrained by one or several EMCs. Each EMC of order l represents the Markov property for a given item variable considered as a continuation of a prefix of length l and is noted $C_{i,l}$.

The probability variable of $C_{i,l}$ is noted $P_{i,l}$. This variable is represented as a finite domain variable (and not as an interval variable), whose domain is the set of probabilities of each continuation at position i given all prefixes of length l:

$$D\left(P_{i,l}\right) = \{P\left(x_i|x_{i-l}, \ldots, x_{i-1}\right)|x_k \in D\left(v_k\right), k = i-l, \ldots, i\}$$

These probabilities are computed once during initialization, by estimating the likelihood of each continuation in the training set. Additionally, each EMC is reified [32], and the associated control variable is noted $B_{i,l}$. When the control variable is set to true, the corresponding EMC is satisfied by n-tuples such that:

$$P_{i,l} = P\left(v_i|v_{i-l}, \ldots, v_{i-1}\right)$$

Control variables may also be used to forbid the use of specific orders (as illustrated in Section 4.1).

In the case of fixed order d, only one EMC of order d is posted on each item variable v_i. In the case of variable order, several EMCs are posted on each item variable v, for each order $l \leq d$ (see Fig. 4). Note that for the first variables v_i with $i \leq$ d only EMCs of order i are posted.

The cost function is defined by a *Sum* constraint on the individual $cost(v_i)$ for each item variable. Individual costs are themselves defined by constraints holding on the probability variables of each EMC holding on v_i as follows:

Fixed-order: $cost_{fixed}\left(v_i\right) = P_{i,d}$

Variable-order/smoothing: $cost_{smooth}\left(v_i\right) = \log\left(\frac{\sum_{l=1}^{d} p_{i,l}}{d}\right)$

This cost is represented by a combination of Log, Sum and Divide constraints holding on the probability variables.

Variable-order/max-order: $cost_{max}\left(v_i\right) = \log\left(P_{i,lmax}\right)$

This cost is represented by an *ad hoc* constraint holding on the probability variables $P_{i,1}, \ldots, P_{i,d}$ and the control variables $B_{i,1}, \ldots, B_{i,d}$. This constraint maintains the value *lmax* in a constrained variable $lmax_i$ whose domain is $\{1, \ldots, d\}$.

Variable-order/algebraic: $cost_{alg}\left(v_i\right) = lmax^2$

This cost is represented by a *Square* constraint holding on the same $lmax_i$.

Arbitrary control constraints can be added to the problem definition. Generated sequences are solutions of the CSP optimizing the cost function. Section 4.1 illustrates how the choice of a given cost function impacts the solutions on a concrete example.

3.4 Propagators for elementary Markov constraints

A solution to implement EMCs could be to use regular constraints as defined in [27]. This would necessitate the construction of the deterministic finite-state automaton corresponding to the language consisting of all subsequences of fixed

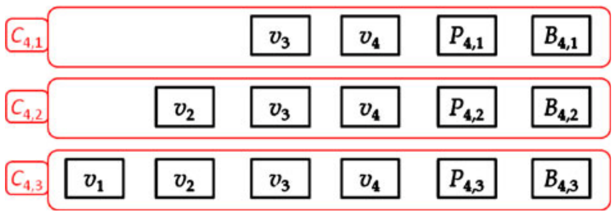


Fig. 4 A stack of EMCs of orders 1 to 3 with variable v_4 as continuation. Each EMC $C_{i,l}$ maintains its probability variable $P_{i,l}$ and control variable $B_{i,l}$

length generated from the training set. Such an automaton can be produced for instance with tools borrowed from string matching, such as the Factor Oracle [2]. Note that such an automaton has been used for building music interactive systems using a random walk approach in the spirit of the Continuator [11].

However, this solution is costly because regular constraints maintain a *layered directed multi-graph* representation of the underlying automaton. This graph contains support information for each couple variable/value, and is updated after each domain modification. It is consequently also saved and restored at each backtrack. The size of this support graph is linearly dependent on the size of the underlying automaton. The complexity of the regular constraint is also directly dependent on the size of this graph. This makes this approach well-suited for regular languages yielding a compact automaton (small number of states). In our case, however, the automaton corresponding to the "language" of fixed order subsequences of the training data can be as large as $|\Sigma| \times d$.

The cost-regular constraint [12] as well as multi-cost [20] extend *regular* by maintaining a cost variable. However, these costs are set on the variable/value pairs, whereas we need to represent the *probabilities* of a continuation given all contexts. More precisely, the costs in a cost-regular constraint with $d + 1$ decision variables are represented by a matrix of dimension $(d + 1) \times |\Sigma|$. For each EMC, we need to consider $|\Sigma|^{d+1}$ probabilities, i.e. one for each tuple.

We propose here a simpler implementation of EMCs, also based on the maintenance of a support list for each variable/value pair. However, we exploit two important observations. Firstly, we know explicitly the set of all sequences (the training set), and therefore all subsequences of interest. Secondly, there is a one-to-one mapping between the tuples that satisfy the constraint and the set of admissible subsequences for the constraint. This mapping can be exploited to represent supports efficiently as explained below.

Following Pesant [27], each EMC maintains a structure that stores, for each item variable v_i and for each value $x \in \Sigma$, the set of all *supports*, i.e. the subsequences supporting the variable-value pair (v_i, x). Additionally, each EMC maintains the list of supports for the probability variable *prob*, i.e. for each $p \in D\ (prob)$ the support of $(prob, p)$ is the set:

$$supports\ (prob, p) = \{(s_1, \ldots, s_{d+1})\,|\,P\,(s_{d+1}|s_1, \ldots, s_d) = p\}$$

During the resolution, each variable v can undergo two domain reduction events: (1) the domain is reduced to a single value, i.e. the variable is instantiated (valueChanged), and (2) a value is removed from the domain of v (valueRemoved). After these events, two main tasks must be performed: (1) maintaining domain consistency for each EMC, and (2) maintaining the support lists.

3.4.1 Management of supports and domain-consistency

For a given EMC C of order d on item variables $(v_1, v_2, \ldots, v_d, v_d + 1)$ and on probability variable *prob*, a support for (v_i, x) is any subsequence $s = s_1, \ldots s_i, \ldots, s_d, s_{d+1}$ with $s_i = x$. A support for $(prob, p)$ is any subsequence $s = s_1, \ldots, s_d, s_{d+1}$ such that $p = P\,(s_{d+1}|s_1, \ldots, s_d)$

We show that an EMC is domain consistent if, and only if, the following property P holds for all $i, j = 1, \ldots, d+1$:

$$\text{(P)}\ \cup_{x\in D(v_i)}\ supports\,(v_i, x) = \cup_{x\in D\left(v_j\right)} supports\ \left(v_j, x\right) = \cup_{p\in D(prob)} supports\,(prob\,,\,p)$$

(P) means that every support for a given variable/value pair is also a support for every other variable, including *prob*, for some value. Domain-consistency of C implies trivially (P). Conversely, (P) entails domain consistency, since:

$$\begin{aligned}
&\forall x\ \in\ D = (v_i)\ such\ that\ supports\ (v_i, x)\ \neq \emptyset\\
&\quad\rightarrow \exists s = (s_1, \ldots, s_{d+1}) \in\ supports\ (v_i, x)\,, i.e.\ s_i = x\\
&\quad\rightarrow \forall j;\ s\ \in\ supports\ \left(v_j, s_j\right)\ and\ \exists p \in\ D\,(prob)\ such\ that s\ \in\ supports\ (prob, p)\\
&\qquad\quad \text{i.e. } p\,(s_{d+1}|s_1, \ldots, d_d) = p\\
&\quad\rightarrow s\ satisfies\ C\\
&\quad\rightarrow x\ is\ consistent\ with\ v_i
\end{aligned}$$

The supports are represented as unique indices in the initial sequences. This light-weight representation is efficient in practice as: (1) the propagators are implemented with constant-time elementary remove operations; and (2) the save and restore operations, performed upon backtracking, can be implemented efficiently (instead of saving/restoring the whole graph).

Ad hoc structures like factor oracles could lead to more compact and efficient representations of the sequences than our explicit representation. However, our representation has a number of advantages. First, it allows to implement efficient propagators for "value changed" and "value removed" events as described below. Additionally, individual sequence lookup is required only once to initialize the problem.

3.4.2 Propagators

The propagators presented here achieve domain consistency as they restore (P) after domain reduction events and do not remove any consistent supports or values.

The support lists, as well as the domains of probability variables are initialized once before the resolution, by procedure init, computed for each EMC. Supports lists are updated during computation and are saved and restored upon backtracking.

```
init (C_{i,l})
    $\forall s = s_1,..,s_l,s_{l+1} \in S^{l+1}$ // The subsequences of length l+1
        $p = P(s_{l+1}|s_1,\ldots,s_l)$
        $D(p_{i,l}) \leftarrow D(p_{i,l}) \cup \{p\}$
        $supports(prob,p) \leftarrow \{s\} \cup supports(prob,p)$
        $\forall k = 1,\ldots,l+1$
            $supports(v_k,\mathrm{s_k}) \leftarrow \{s\} \cup supports(v_k,\mathrm{s_k})$
```

The propagators are the following:

```
valueChanged(var, value) // var was instantiated with value
    $\forall v \neq var$
        $\forall x \in D(v)$
            // keep only supports also supporting value for var
            $supports(v,x) \leftarrow supports(v,x) \cap supports(var, value)$
            if $supports(v,x) = \emptyset$
                // domain-consistency: unsupported values are removed
                $D(v) \leftarrow D(v) \setminus \{x\}$

valueRemoved(var, value) // value was removed from the domain of var
    $\forall v \neq var$
        $\forall x \in D(v)$
            // remove supports of var/value
            $supports(v,x) \leftarrow supports(v,x) \setminus supports(var, value)$
            if $supports(v,x) = \emptyset$
                // domain-consistency: unsupported values are removed
                $D(v) \leftarrow D(v) \setminus \{x\}$
```

The worst-case complexity of these two methods is $O(N \times |\Sigma| \times \sigma)$, where σ is the maximum size of support lists. This is not directly comparable to that of Pesant, which is expressed in terms of a multi-graph structure. It is, however, also linear in the size of the sequence. The initialisation algorithm (init) requires an exhaustive exploration of the training set S but is done only once.

3.5 Entailment of control variables

In the case of variable-order, we introduce approximately $N \times d$ control variables. These variables increase the size of the search space. However, they are not independent since all EMCs with the same continuation are logically related as follows:

$$\begin{gathered} B_{i,l} \rightarrow B_{i,l-1} \\ \textit{and conversely} \\ \textit{not } B_{i,l} \rightarrow \textit{not } B_{i,l+1}. \end{gathered}$$

We represent these entailment relations as *ad hoc* entailment constraints, posted on all pairs of related control variables. This entailment reduces the number of propagated EMCs at any moment, hence the number of filtering operations. Moreover, this entailment speeds up the estimation of the Markovian cost function by the branch and bound process, as it propagates probability variables without any filtering operations.

3.6 Putting it all together

3.6.1 Chunkwise generation

Although the methods we have proposed for filtering Markov constraints are relatively efficient, the process we have described here does not scale up well with the size of the sequence or the maximum order. However, in our context, this is not necessarily an issue. Because we target interactive applications, the sequence length is typically kept small, since the generated sequence is rendered within small time frames. In the music generation example, a reasonable window size is one beat, that is 500 ms at 120 bpm, containing typically between one (quarter note) and eight (32th) notes.

We therefore target a chunk-by chunk generation process, in which each chunk is generated using a CSP as defined in Section 3.3. The generated chunk is rendered, and the process starts again with a new chunk, appended to the preceding one (see Fig. 5). Tiling between chunks is obtained by considering the last chunk as a series of instantiated constrained variables. Chunk size can be determined dynamically with no incidence on the process.

3.6.2 Viewpoints

Another important aspect when dealing with the practical use of Markov models is the viewpoint issue. In our examples, we have considered a single *pitch* or *chord* viewpoint. In practice, items to generate are often more complex. In the case of music, notes have several attributes, including the pitch, but also the duration, velocity, as well as so-called "derived" viewpoints such as intervals, harmonic degrees or intervals thereof, etc. In the case of a graphical application, a line object may be defined by its length, color, width, angle, etc. To represent the Markovian structure of sequences, a choice must be made concerning the dimensions that are taken into account in the Markov model. These dimensions, often referred to as viewpoints [10] play a crucial role, and also raise issues, notably concerning the combination of viewpoints. Our approach does not make any assumption concerning the identification and management of viewpoints, and is compatible with any of these strategies, e.g.:

1. Consider a unique, cross-alphabet viewpoint. The size is roughly that of the Cartesian product of the viewpoint spaces. In this approach, a single viewpoint is considered, like in our example. This approach is likely to produce high-quality solutions but requires a huge training set, so a good compromise must be found concerning the various dimensions considered.
2. Consider each viewpoint separately and define a cost function for each. Then solve the multi-criterion optimization problem by optimizing, e.g. a linear combination of the cost functions.

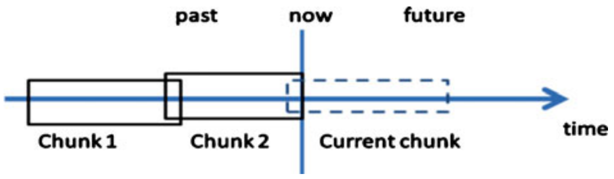


Fig. 5 A chunkwise scenario for infinite sequence generation

Additionally, it is important to note that control constraints may be of different types, depending on the nature of the viewpoint. We can roughly distinguish control constraints that are:

– *Viewpoint dependent*. Typically, equality or difference constraints are posted on sequence variables as we have defined here, i.e. whose domains contain the viewpoint values. For instance, first pitch = last pitch in our example.
– *Viewpoint independent*. Consider the generation of a musical sequence in which notes have also, say *duration* attributes. A constraint could be imposed on the total sum of the durations, to ensure that the resulting sequence has a given total duration. If duration is not used as a viewpoint for individual notes (which is the case in our working example), it is impossible to post such a constraint. However, this situation can be simply handled by adding a series of *item-variables* whose domains are the actual items to be generated (and not their viewpoints), and link these variables to the viewpoint variables as introduced here, through functional constraints. These functional constraints implement a projection from the item domain to the viewpoint domains at a low filtering cost.

4 Validation

In this section we validate our approach in terms of expressiveness (cost functions) and scalability.

4.1 Cost functions

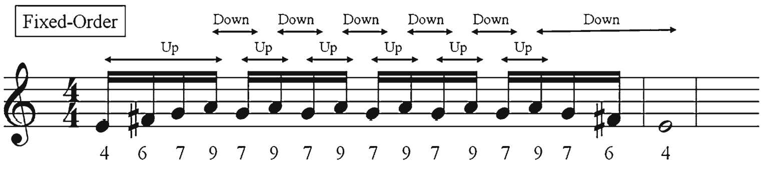
The approach we propose can accommodate for different cost functions, corresponding to the various generation strategies used in practice. Since there is no general rule concerning the choice of the cost function, we illustrate here on a simple problem the possibilities offered by each of them. We consider the problem described in Section 2.2, i.e. the generation of a 17 note melody, given the Al Di Meola training set, and with two anchor constraints (first and last notes equal to E_4, represented here as 4). We show four different optimal solutions for each cost function (note that examples given in Section 2.2 were not optimal solutions). Table 6 shows the respective costs for each solution, and for each cost function. It can be seen that these optima are indeed different (in this case), which emphasizes the importance of the cost function. Section 5.1 shows an application with yet another cost function that takes into account user actions.

Table 6 Costs for each solution (in column), and for each cost function (row)

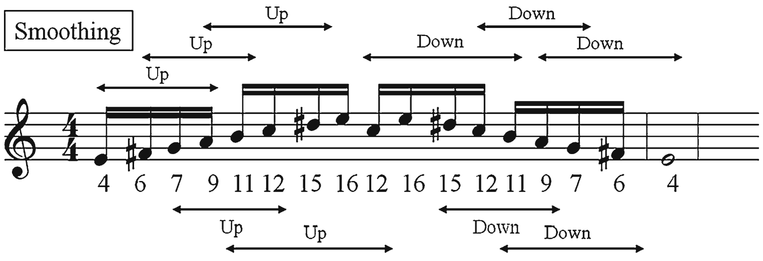
	Fixed order	Smoothed	Max order	Algebraic
Cost fixed order	−13.9	−18.4	−24.7	−22
Cost smoothed	−12.3	−11.3	−11.8	−12.7
Cost max order	−11.1	−6.9	−3.5	−8
Cost algebraic	38	90	120	133

It can be seen that optimum costs are indeed different in this case (i.e. diagonal figures are always strictly maximum in each row)

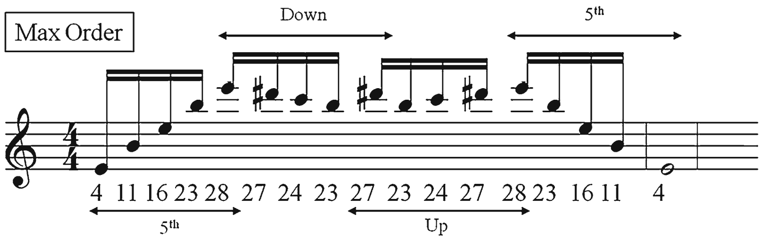
Fixed-order 1:



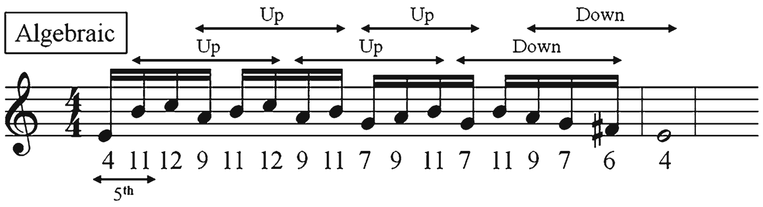
Note that there is no solution to this problem with a fixed-order greater than 1. Variable-order 4 with the smoothing cost function, and order 5 disallowed:



Variable-order 4 with the Max-Order cost function, and order 5 disallowed:



Variable-order 4 with the Algebraic cost function, and order 5 disallowed:



4.2 Scalability

In this section we evaluate the scalability of our approach on an interactive melody generation application. We consider the problem described in Section 2.2: generation of a melody considering the training set of Al Di Meola scales. The control constraints are that the first and last notes are set to E_3. We give here the performance of the approach for finding the optimal solution and the proof of optimality, as a function of the melody length. Domain size for item variables is 15.

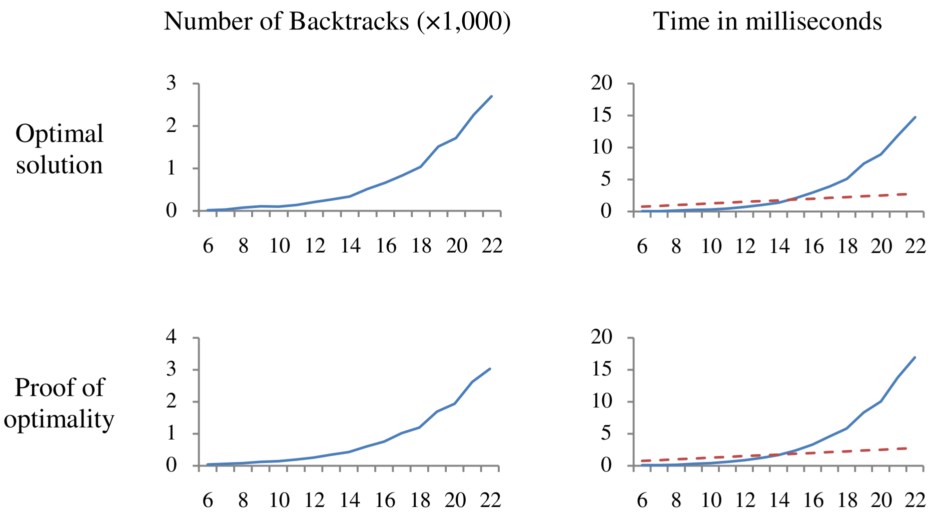


Fig. 6 The performance of our approach on a melody generation problem, compared to the real-time requirements. The *straight line* represents the resolution time (in milliseconds) and the *dotted line* represents the available time in an interaction context, with a tempo of 120 bpm

We compare this performance with a realistic "real-time" requirement. Figure 6 shows that melodies of length up to 14 can be generated within these real-time requirements, on a non-optimized Java prototype.

This experiment shows that our initial real-time requirements can be met in a reasonable context, notwithstanding the various optimizations that can be added to our approach.

5 Applications

5.1 Gesture-based melody composition

5.1.1 Pitch contour as control constraints

In this section we show how our approach can be used to implement a simple gesture-controlled melody generation system. In this case, the task is to generate a melody using the training base described in Section 2. The sampling of a user gesture determines a *pitch contour* for the sequence noted g_i, $i = 1, \ldots N$.

From a CSP viewpoint, the system generates a sequence of 17 notes (representing one full measure of 16th notes plus a full note), which minimizes the distance to the gesture. This distance is itself defined as the sum of the squares of the pitch difference at each position (see Fig. 7). Consequently the cost function is defined as a linear combination of this pitch contour distance with and a "smoothing" Markovian cost:

$$cost\,(s) = a \times cost_{smooth}(s) + (1 - a) \times \sum\nolimits_{i=1}^{N} (s_i - g_i)^2\ ,\ \text{with}\ 0 \leq \alpha \leq 1$$

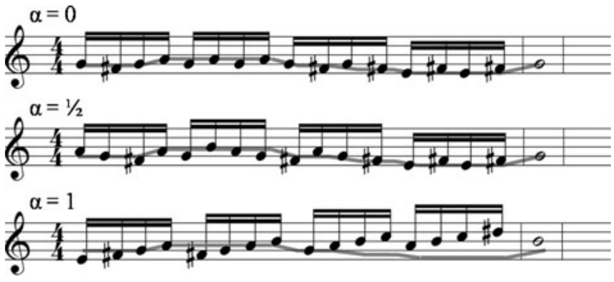


Fig. 7 A user gesture is sampled (*top*) to yield a pitch contour as control constraints. Depending on the value of α solutions are more or less Markovian/close to the pitch contour

The value of α can be in turn controlled by a slider to put more emphasis on the Markovian dimension or the pitch contour. The database is the one presented in Section 2.

5.2 Other applications

Constrained Markov generation has been used for two other interactive applications. The first one is a controllable music improvisation system called Virtuoso. It is basically an extension of the melody generation system, incorporating musical knowledge of Bebop Jazz and Hard-Rock solo improvisation. The user can control in real time an improvisation generated by the system, which fits both with the user gesture and harmonic constraints defined by an arbitrary chord sequence [19].

The second system is a graphical equivalent of the Continuator, called the Doodler. In the Doodler, the user draws freely shapes using the mouse. The Doodler learns a Markov model from user shapes, using information such as the angle between two consecutive lines. The system can then continue an arbitrary line draw by the user. Using our approach, the user can also use the mouse as a control constraint to produce a continuation that is as close as possible to the location of the mouse. More complex constraints, such as a predefined image can also be used as "contour constraints" to force the generation to remain in specific regions of the drawing (see Fig. 8).

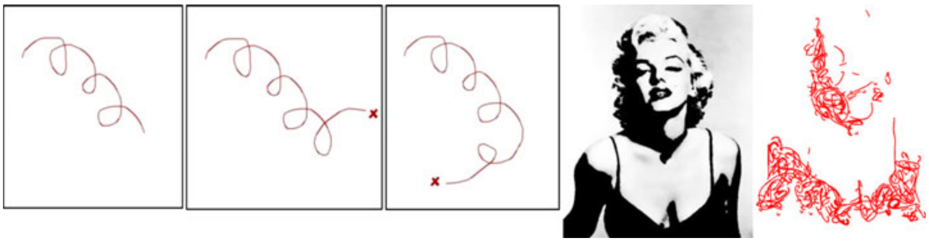


Fig. 8 An application to interactive doodling. First, a shape is drawn by the user (*left*), and the system builds a Markov model from this shape. The user can then click on the screen, and the generator will continue the initial shape with a line sequence that is both Markovian (here, this means the shape is somehow "curly") and whose end point is as close as possible to the clicked point (*red cross*). On the right, a given picture (Marilyn Monroe) is used as a "contour constraint" to force the lines to remain within certain regions

6 Discussion

We have proposed a formulation of Markov sequence generation as a Branch and Bound constraint optimization problem. The Markovian property is represented as a cost function, estimated from a stack of elementary Markov constraints. Such a formulation allows the designer of an interactive Markovian system to post arbitrary additional constraints representing user control criteria, a feature which is impossible with random walk generation. This approach can find sequences whose global probability is very low, therefore which would not be generated, in practice, by greedy approaches.

Our approach can accommodate for arbitrary cost functions, including the cost functions used in practice. As such, our approach solves the problem of steerable Markovian sequence generation in its full generality. We have illustrated our approach on a realistic interactive melody generation system, and on chord sequence generation, two subjects that are traditionally addressed with Markov models.

The scalability of our approach has been shown to be sufficient for real time applications, in which Markov constraints are used chunk-wise in an interactive context. Consequently typical chunk sizes are small, compared to the size of sequences used, e.g. in traditional rostering problems.

However, other optimizations are possible to further reduce the cost of filtering, such as parallel implementation of the Markov constraint, or using more efficient structures to implement the basic filtering operations. Another possibility is to use problem decomposition techniques, such as AND/OR Branch and Bound [18]: because of the local nature of Markov constraints, the problem is not decomposable at creation time, but may become so when adjacent variables are instantiated (unless global constraints are added as control constraints, such as AllDiff or cardinality). Studies of the behavior of cost functions can be conducted to fine-tune variable ordering and value choice heuristics. The purely algebraic approach, which consists in considering only prefixes lengths (and not their probabilities), is also interesting to investigate, notably in relation with variable-length code theory, following [26]. Finally, a promising approach is to use approximation algorithms [17], which produce solutions of cost opt/α, where *opt* is the cost of an optimal solution, and α a fixed ratio. This approach requires the study of the *approximability* of our problem, which could be done given reasonable limitations on the control constraints (such as anchors and binary constraints).

References

1. Addessi, A.-R., & Pachet, F. (2005). Experiments with a musical machine: Musical style replication in 3/5 year old children. *British Journal of Music Education, 22*(1), 21–46.
2. Allauzen, C., Crochemore, M., & Raffinot, M. (1999). Factor oracle: A new structure for pattern matching. In J. Pavelka, G. Tel, & M. Bartosek (Eds.), *Proceedings of the 26th conference on current trends in theory and practice of informatics (November 27–Dec. 4), LNCS* (Vol. 1725, pp. 295–310). London: Springer.

3. Anders, T., & Miranda, E. R. (2008). Constraint-based composition in real time. In *Proc. of int. computer music conference, Belfast, UK*.
4. Anders, T., & Miranda, E. R. (2010). Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys* (in press).
5. Assayag, G., Dubnov, S., & Delerue, O. (1999). Guessing the composer's mind: Applying universal prediction to musical style. In *Proc. of int. computer music conference, Bejing, China, ICMA*.
6. Bejerano, G. (2004). Algorithms for variable length Markov chain modeling. *Bioinformatics, 20*(5), 788–789.
7. Beldiceanu, N., Carlsson, M., & Demassey, S. (2007). Global constraint catalogue: Past, present and future. *Constraints, 12*(1), 32–62
8. Brooks, F. P., Hopkins, A. L., Neumann, P. G., & Wright, W. V. (1993). An experiment in musical composition. In S. M. Schwanauer & D. A. Levitt (Eds.), *Machine models of music* (pp. 23–40). Cambridge: MIT.
9. Conklin, D. (2003). Music generation from statistical models. In *Proc. of AISB 2003 symposium on artificial intelligence and creativity in the arts and sciences* (pp. 30–35).
10. Conklin, D., & Witten, I. H. (1995). Multiple viewpoint systems for music prediction. *Journal of New Music Research, 24*(1), 51–73.
11. Cont, A., Dubnov, S., & Assayag, G. (2007). Anticipatory model of musical style imitation using collaborative and competitive reinforcement learning. In *Anticipatory behavior in adaptive learning systems. LNCS* (Vol. 4520/2007, pp. 285–306). Berlin: Springer.
12. Demassey, S., Pesant, G., & Rousseau, L.-M. (2006). A cost-regular based hybrid column generation approach. *Constraints, 11*(4), 315–333.
13. Di Meola, A., McLaughlin, J., & De Lucia, P. (1992). *Friday night in San Francisco*. Milwaukee: Hal Leonard Corporation.
14. Eigenfield, A., & Pasquier, P. (2010). Realtime generation of harmonic progressions using controlled markov selection. In *Proc. of the 1st international conference on computational creativity (ICCCX)* (pp. 16–25). Lisbon: ACM.
15. Farbood, M., & Schoner, B. (2001). Analysis and synthesis of palestrina-style counterpoint using Markov chains. In *Proc. of international computer music conference, Cuba* (pp. 471–474).
16. Ilog (2001) Ilog solver 5.2 reference manual. Available online at https://www.enseignement.polytechnique.fr/local/documentation/ilog/solver52/doc/refman/html/IlcTableConstraint.html.
17. Johnson, D. S. (1974). Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences, 9*(3), 256–278.
18. Marinescu, R., & Dechter, R. (2005). Advances in AND/OR branch-and-bound search for constraint optimization. In *The 7th international workshop on preferences and soft constraints of the eleventh international conference on principles and practice of constraint programming, CP'2005*.
19. Markov Constraints (2010). Available online at http://www.csl.sony.fr/MarkovCt. Accessed September 2010.
20. Menana, J., & Demassey, S. (2009). Sequencing and counting with the multicost-regular constraint. In *Lecture notes in computer science, 6th international conference on integration of AI and OR techniques in constraint programming for combinatorial optimization problems (CPAIOR'09), Pittsburgh, USA*.
21. Nierhaus, G. (2009). *Algorithmic composition—paradigms of automated music generation*. Berlin: Springer.
22. Pachet, F. (2002). The continuator: Musical interaction with style. In *Proc. of international computer music conference* (pp. 211–218). Goteborg, Sweden. September, Best Paper Award.
23. Pachet, F., Roy, P., & Cazaly, D. (2000). A combinatorial approach to content-based music selection. *IEEE Multimedia, 7*(1), 44–51.
24. Parker, C. (2001). *Charlie Parker omnibook: For C instruments*. Van Nuys: Alfred Publishing.
25. Pearce, M., & Wiggins, G. (2004). Improved methods for statistical modeling of monophonic music. *Journal of New Music Research, 33*(4), 367–385.
26. Perrot, J.-F. (1977). Informatique et Algèbre, La théorie des codes à longueur variable. In *Theoretical computer science, G. I. conference, lecture notes in computer science* (3rd Ed., no. 48, pp. 27–44).
27. Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. *Lecture Notes in Computer Science, 32*(58), 482–495.
28. Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*(2), 257–286.

29. Radicioni, D. P., & Lombardo, V. (2007). A constraint-based approach for annotating music scores with gestural information. *Constraints, 12*(4), 405–428.
30. Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proc. of the 12th national conference on artificial intelligence. AAAI 1994* (pp. 362–367).
31. Ron, D., Singer, Y., & Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning, 25*(2–3), 117–149.
32. Rossi, F., van Beek, P., & Walsh, T. (Eds.) (2006). *Handbook of constraint programming*. New York: Elsevier.
33. Simhon, S., & Dudek, G. (2004). Sketch interpretation and refinement using statistical models. In H. W. Jensen, & A. Keller (Eds.), *Eurographics symposium on rendering*.
34. Steedman, M. J. (1984). A generative grammar for jazz chord sequences. *Music Perception, 2*(1), 52–77.
35. van Hoeve, W. J., & Katriel, I. (2006). *Global constraints. Handbook of constraint programming* (chap. 6). New York: Elsevier Science Inc.
36. van Hoeve, W.-J., Pesant, G., Rousseau, L.-M., & Sabharwal, A. (2006). Revisiting the sequence constraint. In *Proc. of the 12th international conference on principles and practice of constraint programming (CP 2006). LNCS 4204* (pp. 620–634).
37. Witten, I. H., & Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transaction on Information Theory, 37*(4), 1085–1094.