In proc. of *International Computer music Conference*, Gotheborg (Sweden), ICMA, September 2002.

The Continuator: Musical Interaction With Style

François Pachet
Sony Computer Science Laboratory
6, rue Amyot, Paris 75005
pachet@csl.sony.fr
http://www.csl.sony.fr/~pachet

citer zicarelli (M and Jam Factory, CMJ 1987) et David Wessel et JC Risset (CMJ) et Lartillot et al. 01

Abstract

We propose a system, the Continuator, that bridges the gap between two classes of traditionally incompatible musical systems: 1) interactive musical systems, limited in their ability to generate stylistically consistent material, and 2) music imitation systems, which are fundamentally not interactive. Our purpose is to allow musicians to extend their technical ability with stylistically consistent, automatically learnt material. This goal requires the ability for the system to build operational representations of musical styles in a real time context. Our approach is based on a Markov model of musical styles augmented to account for musical issues such as management of rhythm, beat, harmony, and imprecision. The resulting system is able to learn and generate music in any style, either in standalone mode, as continuations of musician's input, or as interactive improvisation back up. Lastly, the very design of the system makes possible new modes of musical collaborative playing. We describe the architecture, implementation issues and experimentations conducted with the system in several real world contexts.

1. Introduction

Music improvisation is both a fascinating activity and a very frustrating one. Playing music requires an intimate relationship between musical thought and sensory-motor processes: the musician must think, listen, develop ideas and move his/her fingers very quickly. The speed and lack of time is a crucial ingredient of improvisation; it is what makes it exciting. It is also what makes it frustrating: beginners as well as experienced musical performers are by definition limited by their technical abilities, and by the morphology of the instrument.

We propose to design musical instruments that address explicitly this issue: providing real time, efficient and enhanced means of generating *interesting* musical material.

Musical performance has been the object of numerous studies, approaches and prototypes, using virtually all the computer techniques at hand. In our context, we can divide these approaches in two categories: *interactive systems* and music *imitation* systems. Schematically, interactive music systems propose ways of transforming quickly musical input into musical output. Musical interactive systems have been popular both in the experimental field (Baggi, 1992), (Biles, 1998) as well as in commercial applications, from one-touch chords of arranger systems to the recent and popular Korg Karma synthesizer (Karma, 2001). While a lot of work has been devoted to efficient controllers and interfaces for musical systems (Borchers, 1999), (Nime, 2001), these systems all share a common drawback: they are not able to learn, there is no memory of the past. Consequently the music generated is strongly correlated with musical input, but not or poorly with a consistent and realistic musical style.

On the other hand, music imitation systems precisely aim at representing stylistic information, to generate music in various styles: from the pioneering Illiac suite by Hiller and Isaacson (1959) to the automatic compositions of (Cope, 1996). More recently, constraint techniques have been used to produce stylistically consistent 4-part Baroque music (see (Pachet & Roy, 2001) for a survey). In the domain of popular music, prototypes such as (Biles, 1998) or (Ramalho, 1994) have demonstrated the technical feasibility of simulating convincingly jazz styles by computer. These systems propose fully-fledged automata that may produce impressively realistic music, but they do not support musical interaction and cannot be used as actual instruments. Moreover, these approaches require explicit, symbolic information to be fed to the system, such as human input for supervised learning, underlying harmonic structure, tempo, song structure, which further limits their usability.

The system we present here is an attempt to combine both worlds: real-time interactive musical instruments that are able to produce stylistically consistent music.

More precisely, we propose a system in which musical styles are learned automatically, in an agnostic manner, and therefore do not require any symbolic information (style, harmonic grid, tempo). The system is seamlessly integrated in the playing mode of the musician, as opposed to traditional question/answer or fully automatic systems, and adapts quickly and without human

intervention to unexpected changes in rhythm, harmony or style. Finally, the very design of the system allows the sharing of stylistic patterns in real time and constitutes in this sense a novel form of collaborative musical instrument.

The remaining of the paper is structured as follows. First we introduce the architecture of the proposed system, its inputs and outputs. We then describe the heart of the engine, based on a Markov based model of musical styles. This model is augmented with 1) a hierarchical model of learning functions to adapt to imprecision in musical inputs and 2) a facility for biasing the Markovian generation, to handle external information such as changing harmony. Finally, we illustrate the use of the system in various musical contexts: solos, accompaniments, and collaborative music improvisation.

2. Architecture

In this paper we focus on a Midi system linked to an arbitrary midi controller. Experiments described here were conducted with Midi keyboard and guitars, and are easily applicable to any style and Midi controller. An audio version is currently in progress, and the ideas proposed in this paper are in a large respect independent of the nature of the information managed.

We consider music as temporal sequences of Midi events. The information we represent are: pitch (integer between 0 and 127), velocity/amplitude (also between 0 and 127), and temporal information on start and duration times, expressed as long integers, with a precision of 1 millisecond, as provided by the MidiShare Midi operating system (Orlarey & Lequay., 1989). In the standard playing mode, the system receives input by one musician. The output of the system is sent to a Midi synthesizer and then to a sound reproduction system (see Figure 1).

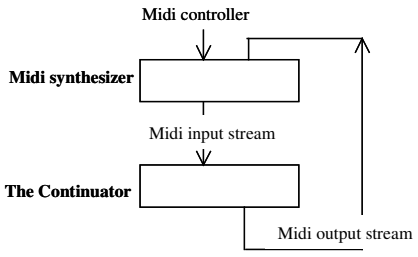


Figure 1. Flow of information to and from the Continuator.

The system acts basically as a sequence continuator: the note stream of the musician is systematically segmented into phrases using a variable temporal threshold (typically about 250 milliseconds). Each phrase is sent asynchronously to a phrase analyzer, which builds up a model of recurring patterns. In reaction to the played phrase, the system generates a new phrase, built as a continuation of the input phrase, according to the database of patterns already learnt.

In the next section we describe the heart of the system, based on an extension of a Markov model.

3. Learning musical style, pratically

Markov chains and music is an old and rather repetitive story. The most spectacular application to music is probably the compositions of (Cope, 1996), whose system is able to represent faithfully musical styles. However, his ad hoc scheme is not easily reproducible and extensible. One major interest of Markov-based models is that they are naturally able to generate new musical material in the style learned. Recently, variations of the basic Markov models have been introduced to improve the efficiency of the learning methods, as well as the accuracy of the music generated (Assayag et al., 1999), (Lartillot et al, 2001), (Trivino-Rodrigues & Morales-Bueno, 1999). In all cases, the main idea is to capture the local patterns found in the learnt corpus, using probabilistic schemes. New sequences are then generated using these probabilities. These sequences will contain, by construction, the patterns identified in the learnt corpus.

These works show clearly two things: 1) Markov chain models (and their extensions, notably for variable-length) are able to represent efficiently musical patterns, but 2) there generative power is limited due to the absence of long-term information. In another words, these models can fool the listener on a short scale, but not for complete pieces. Using Markov models for interaction purposes allows us to benefit from 1) and avoid the drawback of 2). The responsibility for organizing the piece, deciding its structure, etc. are left to the musician. The system only "fills in the gaps", and therefore the power of Markov chain can be exploited fully.

We address in the following sections the main issues involved in building effective and realistic models of musical styles:

- Efficiency and the ability to perform the learning in real time,
- A realistic management of continuity,
- The handling of specifically musical issues such as rhythm and polyphony.

3.1. Learning sequences efficiently

The learning module we propose systematically learns all phrases played by the musician, and progressively builds up a database of patterns detected in the input sequences. After initial experiments with incomplete learning schemes such as the Lempel-Ziv mechanism described in (Assayag et al., 1999), we designed an indexing scheme which represents all the subsequences found in the corpus, in such a way that the computation of continuations is 1)

complete and 2) as efficient as possible. We describe here briefly the design of this learning scheme, which can be seen as an efficient implementation of a complete variable-order Markov model of input sequences, as initially introduced by (Ron et al., 1996).

This technique consists in building a prefix tree by a simple, linear analysis of each input sequence. Each time a sequence is input to the system, it is parsed from right to left and new prefixes encountered are systematically added to the tree. Each node of the tree is labeled by a *reduction function* of the corresponding element of the input sequence. In the simplest case, the reduction function can be the pitch of the corresponding note. We describe in the next section more advanced reduction functions, and stress on the their role in the learning process. To each tree node is attached a list of continuations encountered in the corpus. These continuations are represented as integers, denoting the index of the continuation item in the input sequence. This indexing scheme makes it possible to avoid duplicating data by manipulating only indexes. When a new continuation is found for a given node, the corresponding index is added to the node's continuation list (shown in the figure between accolades {}).

For instance, suppose the first input sequence is {A B C D}. We will progressively build the tree structure illustrated in Figure 2. These trees represent all possible prefixes found in the learnt sequences, in reverse order, to facilitate the generation process (see next section). In the first iteration, the sequence is parsed from right to left, and produces the left tree of Figure 2. First, the node C is created, with continuation index {4}, representing the last D of the input sequence. Then node B is added as a son of node C, with the same continuation index {4}. Finally, node A is created as a son of node B, with the same continuation index.

Then the parsing starts again for the input sequence minus its last element, i.e. {A B C}, to produce the middle tree of Figure 2. In this tree, all nodes have {3} as a continuation (meaning item C). Finally, the sequence {A B} is parsed and produces the tree on the right of Figure 2. Nodes are created only once the first time they are needed, with empty continuation lists. The tree grows as new sequences are parsed, initially very quickly, then more slowly as patterns encountered are repeating.

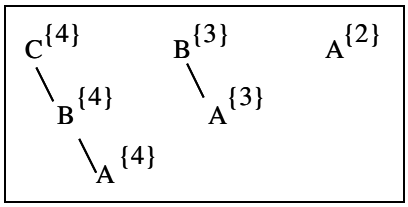


Figure 2. The tree of the patterns found in {A B C D }

Now, let us see what happens when the second following sequence is parsed: { A B B C}.

Using the same mechanism, we parse again the input sequence from right to left, to produce the continuation index 8 (i.e. C) for {A B C}. We get the following updated tree structure, where the new nodes and continuations are indicated in red:

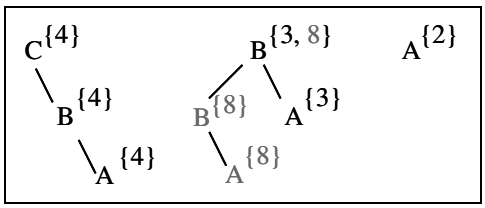


Figure 3. The tree structure augmented with the parsing of { A B B C }.

We keep on parsing with the truncated subsequences until we get the following graph (Figure 4):

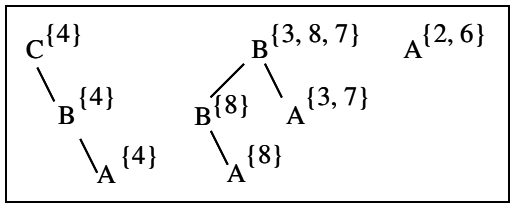


Figure 4. The complete graph corresponding to sequences {A B C D} and {A B B C}.

As we will see below, this graph has the property that retrieving continuations for any subsequence is extremely fast, and requires a simple walkthrough the input sequence.

3.2. Generation of continuations

The second module of our system is the real time continuation mechanism, which generates the music in reaction to an input sequence. The generation is performed using a traversal of the trees built from input sequences. The main property of this generation is that it produces sequences which are locally maximally consistent, and which have the same Markovian distributions.

The generation is performed by producing items one by one, and, at each iteration, considering the longest possible subsequence. Once a continuation is generated, the process is repeated with the input sequence augmented by the continuation. This tiling mechanism makes the real time generation possible, as we will see in the next sections. This process, referred to as variable-order Markov chains is the following. Suppose an input sequence such as:

{A B }

We walk through the previously built tree to look for all continuations of {A B}. We start by looking for a root node corresponding to the last element of the input sequence (B). We then walk down this tree to match the input sequence until we either complete the input

sequence, or do not find the corresponding node. When the walkthrough is finished, we simply return the set of continuations of the corresponding node. In our case we find a continuation for the whole input sequence {A B}:

Continuation_List ({A B}) = {3, 7}.

Theses indexes correspond to items {C, B}. A continuation is then chosen by a random draw. Suppose we draw B. We then start again with the new sequence {A B B}, for which we repeat the retrieving process to find the continuation list:

Continuation_List ({A B B }) = {8}.

We chose the only possible continuation (index 8 corresponds to item C) and get {A B B C}. We do not find any continuation for the whole sequence {A B B C}, but we get continuations for the longest possible subsequence, that is here:

Continuation_List ({ B C}) = {4}.

We therefore get the sequence {A B B C D} and continue the generation process. At this point, there is no continuation for {A B B C D} as well as for any subsequence ending by D (indeed, D has always been a terminal item in our learnt corpus).

In this case, when no continuation is found for the input sequence, a node is chosen at random. We will see in the next section a more satisfactory mechanism for handling such cases of discontinuity.

It is important to note that, at each iteration, the continuation is chosen by a random draw, weighted by the probabilities of each possible continuation. The probability of each continuation is directly given by drawing an item with an equal probability distribution, since repeating items are repeated in the continuation list. More precisely, for a continuation x, its probability is:

Markov_Prob(x) = nb of occurrences of x in L, where *L* is the continuation list.

Since the continuations are in fact indexes to the original sequences, the generation can use any information from the original sequence which is not necessarily present in the reduction function (e.g. velocity, rhythm, midi controllers, etc.): the reduction function is only used to build the tree structure, and not for the generation *per se*.

4. Reduction functions

As we saw in the preceding section, the graph is not built from raw data. A Midi sequence has many parameters, all of which are not necessarily interesting to learn. For instance, a note has attributes such as pitch, velocity, duration, start time. A chord has attributes such as the pitch list, possibly its root key, etc. The system we propose allows the user to choose explicitly from a library of predefined reduction functions. The simplest function is the pitch. A more refined function is the combination of pitch and duration. Trivino-Rodrigues & Morales-Bueno (2001) introduced the idea of multi-attribute Markov models for learning musical data, and made the case that handling all attributes requires in principles a Cartesian product of attribute domains, leading to an exponential growth of the tree structures. The model they propose allows to avoid building the Cartesian product, but does not take into account any form of imprecision in input data. Conklin & Witten (1995) propose different reduction functions (called viewpoints) for representing music. Our experiments with real music led us to develop and implement such a library of reduction functions, including the ones mentioned in these works, as well as functions specially designed to take into account realistic Jazz styles. One of them is the *PitchRegion*, which is a simplification of pitch. Instead of considering explicitly pitches, we reduce pitches in regions, practically by considering only *pitch / region_size*.

4.1. Hierarchical graphs

One important issue in dealing with Markov models if the management of imprecision. By definition, Markov models deal with perfect strings, and there is no provision for handling imprecision. In our example, the String {A B C X} has no continuation, simply because symbol X has no continuation. In the approaches proposed so far, such case would trigger the drawing of a random node, thereby breaking somehow the continuity of the generated sequence.

The treatment of inexact string matching in a Markovian context is addressed typically by Hidden Markov Models. In this framework, the state of the Markov model are not simply the items of input sequences, as other, hidden state are inferred, precisely to represent state regions, and eventually cope with inexact string inputs. However, Hidden Markov Models are much more complex than Markov models, and are cpu consuming, especially in the generation phase. More importantly, the determination of the hidden states is not controllable, and may be an issue in the practical context we are dealing with here.

We propose here another approach, based on a simple remark. Suppose a model trained to learn the arpeggio in figure 5:

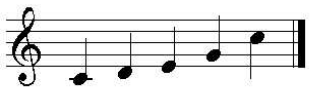


Figure 5. An arpeggio learnt by the Continuator.

Suppose that the reduction function is as precise as possible, say pitch, velocity and duration. Suppose now that the input sequence to continue is the one in Figure 6.

It is clear that any Markov model will consider that there is no continuation for this sequence, simply because there is no continuation for *Eb*. The models proposed so far

would then draw a new note at random, and actually start a new sequence.

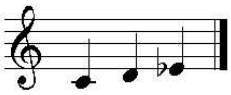


Figure 6. An input sequence which does not match exactly with the learnt corpus.

However, it is also clear intuitively, that a better solution, in such a case is to shift the viewpoint. The idea is to consider a less refined reduction function. In this case, les us consider for instance pitch regions of three notes instead of pitches.

The learnt sequence is then reduced to: {PR1 PR1 PR2 PR3 PR5}

The input sequence is reduced to: {PR1 PR1 PR2}

In this new model, there is a continuation for {PR1 PR1 PR2}, which is PR3.

Because our model keeps track of the index of the data in the input sequences (and not the actual reduction functions), we can generate the note corresponding to PR3, in our case, G. Once this continuation has been found, the process is started again with the new sequence, using the more refined reduction function.

More precisely, we introduce a *hierarchy* of reduction functions, to be used in a certain order in cases of failure. This hierarchy can be defined by the user. Typically, a useful hierarchy can be the following:

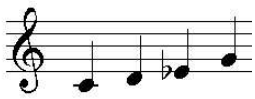
1 – pitch * duration * velocity

2 – small pitch region * velocity

3 – small pitch regions

4 – large pitch regions,

where the numbering indicates the order in which the graphs are considered in cases of failure. In this case, a possible continuation found by our system would be as follows, with an edequate handling of the Eb "imprecision":



The approach we propose allows to take into account inexact inputs, at a minimum cost. The complexity for retrieving the continuations for a given input sequence is indeed very small as it involves only walking through trees, without any actual search.

5. Polyphony and Rhythm

Before describing how we turn our model into a real time interactive system, we have to explain how we handle several important musical issues, which are crucial to ensure that the generation is realistic musically.

5.1. Polyphony

Because our model is based on sequences of discrete data, we have to ensure that the items in the model are in some sort independent, to be recombined safely with each other. With arbitrary polyphony in the input, this is not always the case, as illustrated in Figure 7: some notes may not be stylistically relevant without other notes sounding at the same time.

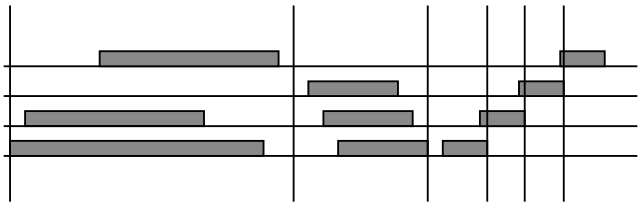


Figure 7. Handling polyphony with segmentation. Chords are clustered (on the left), and legato notes are separated (on the right).

Assayag et al. (1999) propose a scheme for handling polyphony consisting in slicing up the input sequence according to every event boundary occurring in any voice. This scheme is satisfactory in principle, in that it allows to model intricate contrapuntal relationships between several voices. In practice, we experimented with various schemes and came up with a simpler model more fitted with the properties of real interactive music. XXXTaking into account the "negative time"

We first apply an aggregation scheme to the input sequence, in which we aggregate clusters of notes sounding approximately "together". This situation is very frequent in music for instance with the use of pedals. Conversely, to manage legato playing styles, we treat notes slightly overlapping as actually different (see the end of the figure) by considering that an overlap of less than a few milliseconds is only the sign of legato, not of an actual musical cluster.

These cases are actually tricky to handle at the generation phase, because some delay can be introduced, if one simply regenerates the sequence of notes as contiguous. To cope with this situation, the respective inter note delays are memorized and introduced again at the generation phase.

5.2. Rhythm

Rhythm refers to the temporal characteristics of musical events (notes, or clusters). Rhythm is an essential component of style and requires a particular treatment. In our context, we consider in effect that musical sequences are generated step by step, by reconstructing fragments of already parsed sequences. This assumption is unfortunately not always true, as some rhythms do not

afford reconstruction by slicing arbitrarily bits and pieces. As Figure 7 illustrates, the standard clustering process does not take into account the rhythmic structure, and this may lead to strange rhythmical sequences at the generation phase.

This problem has no universal answer, but different solutions according to different musical contexts. Based on our experiments with jazz and popular music musicians, we have come up with three different *modes* that the user can choose from:

Natural rhythm: The rhythm of the generated sequence is the rhythm as it was encountered during the learning phase. In this case, the generation explicitly restitutes the temporal structure as it was learned, and in particular "undoes" the aggregation performed and described in the previous section.

Linear rhythm: this mode consists in generating only streams of eight-note, that is with a fixed duration and all notes concatenated. This allows generating very fast and impressive phrases, and is particularly useful in the be-bop style.

Input rhythm: in this mode, the rhythm of the output is the rhythm of the input phrase, possibly warped if the output is longer than the input. This allows to create continuations that *sound like* imitations rhythmically.

Fixed metrical structure: For popular and heavily rhythmic music, the metrical structure is very important and the preceding modes are not satisfactory. Conklin and Witten (1995) suggest to use the location of a note in a bar as yet another viewpoint, but this scheme forces to use quantization, which in run raises many issues which are intractable in an interactive context.

Instead, we propose in this mode to segment the input sequences according to a fixed metrical structure. The metrical structure is typically given by an external sequencer, together with a given tempo, through Midi synchronization. For instance, it can be 4 beats, with a tempo of 120. In this case (see Figure 8), the segmentation ensures that notes are either truncated at the ending of the temporal unit when they are too long, or at the beginning of the unit if they begin too early.

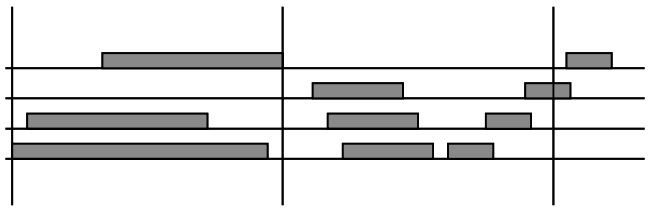


Figure 8. Handling polyphony with fixed segmentation

6. Turning the Generator into an Instrument

The learning and generation module described in the preceding sections are able to generate music sequences that sound like the sequences in the learnt corpus. As such, this provides a powerful musical automaton able to imitate faithfully styles, but not a musical instrument. This section describes the main design concepts that allow to turn this style generator into an interactive musical instrument. This is achieved through two related constructs: 1) a step-by step generation of the music sequences achieved through a real time implementation of the generator, and most importantly 2) a modification of the basic Markovian generation process by the adjunction of a *fitness function* which takes into account characteristics of the input phrase.

6.1. Real time generation

The real time generation is an important aspect of the system since it is precisely what allows to take into account external information quickly, and ensure that the music generated follows accurately the input, and remains controllable by the user. The most important aspect of the real time architecture is that the generation of musical sequences is performed step-by step, in such a way that any external information can be used to influence the generation (see next section). The generation is performed by a specific thread (generation thread), which generates the sequence by chunks. The size of the chunks is parameterized, but can be as small as 1 note event. Once the chunk is generated, the thread sleeps and wakes up for handling the next chunk in time.

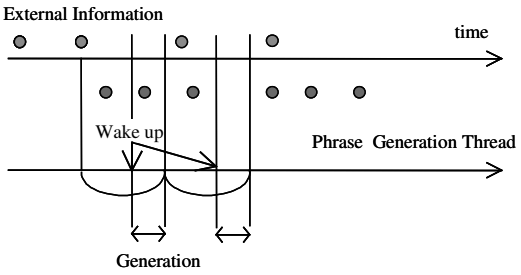


Figure 9. The step-by-step generation process can take into account external information continuously.

6.2. Biasing the Markov Generation

The main idea to turn our automaton into an interactive system is to influence the Markovian generation by real time characteristics of the input. As we saw above, the very idea of Markov-based generation is to produce sequences in such a way that the probabilities of each item of the sequence are the probabilities of occurrences of the items in the learnt corpus.

In the context of musical interaction, this property is not always the right one, because many things can happen during the generation process. In particular, in the case of tonal music, the harmony can change. Typically, in a Jazz trio for instance, the pianist play chords which have no reason to be always the same, throughout the generation process. Because we target a real world performance context, these chords are not predictable, and cannot be learnt by the system prior to the performance. The system should be able somehow to take this external information into account during the generation, and twist the generated sequence in the corresponding directions.

The idea is to introduce a constraint facility in the generation phase. External information may be sent as additional input to the system. This information can be typically the last 8 notes (pitches) played by the pianist for instance, if we want the system to follow harmony. It can also be the velocity information of the whole band, if we want the system to follow the amplitude, or any information that can be used to influence the generation process. This external input is used as follows: when a set of possible continuation nodes is computed (see section on generation), instead of choosing a node according to its Markovian probability, we weight the nodes according to how they match the external input. For instance, we can decide to prefer nodes whose pitch is in the set of external pitches, to favor branches of the tree having common notes with the piano accompaniment.

In this case, the harmonic information is provided implicitly, in real time, by one of the musician (possibly the user himself), without having to explicitly enter the harmonic grid or any symbolic information in the system.

More precisely, we consider a function *Fitness* (x, Context) with value in [0, 1] which represents how well item *x* fits with the current context. For instance, a Fitness function can represent how harmonically close is the continuation with respect to external information. If we suppose that *piano* is the set of the last 8 notes played by the pianist for instance, *Fitness* can be defined as:

$$Fitness(p, piano) = | p \cap piano | \ / \ |piano|$$

This fitness scheme is of course independent of the Markovian probability defined above. We therefore introduce a new weighting scheme which allows to parameterize the importance of the external input, via a parameter *S* (between 0 and 1):

$$Prob(x) = S * Markov\_Prob(x) + (1 - S) * Fitness(x, Context)$$

By setting *S* to extreme values we eventually get two extreme behaviors:

- S = 1, we get a musical automaton insensitive to the musical context,
- S = 0, we get a reactive system which generates the closest musical elements to the external input it finds in the database.

Of course, interesting values are intermediary: when the system generates musical material which is both stylistically consistent, and sensitive to the input. Experiments in these various modes are described below.

7. Experiments

We have conducted a series of experimentations with system, in various modes and configurations. There are basically two aspects we can assess:

1 The musical quality of the music generated,

2 The new collaborative modes the system allows.

We review each of these aspects in the following sections.

7.1. Musical Quality

It is difficult to describe music by words, and rate its quality, especially jazz improvisation. However, we can easily rate how the system differs from the human input. We have conducted tests to check whether listeners could tell when the system is playing or not. In most of the cases, if not all, the music produced is undistinguishable from the user's input. This is typically true for quick and fast solos (keyboard or guitar) in which the system generates long and often impressive jazzy phrases in the style of Pat Martino, John Mc Laughlin, or Alan Holdsworth, depending on the selected input database.

Concerning fixed metrical structure, experiments in the various styles of the Karma music workstation were recorded. In these experiments, we have connected the Continuator to the Korg Karma workstation, both in input and output. The Continuator is used as an additional layer to the Karma effect engine. The Continuator is able to generate infinite variations from simple recordings of music, in virtually all the styles proposed by the Karma. Audio samples can be heard at our web site.

7.2. New Collaborative Music Modes

An interesting consequence of the design of the system is that it leads to several new playing modes with other musicians. Traditionally, improvised music has consisted in quite limited types of interaction, mostly based around question/answer systems (Baggi, 1996), (Walker, 1997). With the Continuator, new musical modes can be envisaged:

- *Single autarcy*. One musician plays with the system after having fed the system with a database of improvisations by a famous musician, as Midi files. We have experimented in particular with a database of midi choruses from Pat Martino, provided by (Heuser, 1994), and a database of Bernard Lubat's piano style.
- *Multiple autarcy*: each musician has its own version of the system, with its own database. This provides a traditional setting in which each musician plays with

his/her own style. Additionally, we experimented improvisations in which one musician (Gyorgy Kurtag) had several copies of the system linked to different midi keyboards. The result for the listener is a dramatic increase in musical density. For the musician, the subjective impression ranges from a "cruise" button with which he only has to start a sequence and let the system continue, to the baffling impression of a musical amplifying mirror.

- *Master/Slave*: one musician uses the system in its basic form, another (e.g. pianist) provides the external data to influence the generation. This is typically useful for extending a player's solo ability while following the harmonic context provided by another musician. Conversely, the system can be used as an automatic accompaniment system which follows the user. In this configuration, the Continuator is given a database of chord sequences, and the input of the user is used as the external data. Chords are played by the system so as to satisfy simultaneously two criteria: 1) continuity, as given by the learnt corpus (e.g. two fives, harmonic cadenzas, etc.) and 2) closeness to the input. The samples show clearly how the user tries to fool the system by playing quick transposition and strange harmonies. In all cases, the Continuator finds chords that match the input as closely as possible. A particularly striking example is a Bach prelude (in C) previously learnt by the system, and used for generation of an infinite stream of arpeggios. When the user plays single chords on a keyboard, the arpeggios instantaneously "follow" the chords played.
- *Cumulative*: all musicians share the same pattern database. This setting was experimented during a Jazz festival (Uzeste, France), where two musicians played with the same (Bernard Lubat) database,
- *Sharing*: each musician plays with the pattern database of the other (e.g.; piano with guitar, etc.). This creates exciting new possibilities as a musician can experience playing with unusual patterns.

8. Conclusion

We have described a music generation system, which is able to produce music satisfying two traditionally incompatible criteria: 1) stylistic consistency and 2) interactivity. This is made possible by introducing several improvements to the basic Markovian generation, and by implementing the generation as a real time, step-by-step process. The resulting system is able to produce musical continuations of any user – including beginners - according to previously learnt, arbitrary styles. Additionally, the design of the system makes it possible to share musical styles, and thus to open new modes of collaborative playing.

Current work is devoted to the elaboration of an extensive style library by recording material from experienced, top-level musicians of various styles (jazz, funk, baroque). An audio version is under progress, in which the input to the system is an audio stream. The stream is analyzed in real time to extract meaningful segments, not necessarily corresponding to actual musical notes. These segments together with basic audio descriptors (pitch, zero crossing rate, energy, etc.) are then fed to the system described here. This will allow using the system for non-Midi instruments, voice in particular.

Acknowledgments

We thank György Kurtag, Bernard Lubat and Alan Silva for intense and fruitful interactions with the system.

References

Assayag, G. Dubnov, S. Delerue, O. Guessing the Composer's Mind: Applying Universal Prediction to Musical Style, Proc. ICMC 99, Beijing, China, 1999.

Baggi, D. L. NeurSwing: An Intelligent Workbench for the Investigation of Swing in Jazz, in Readings in Computer Generated Music, IEEE Computer Society Press, 1992.

Biles, John A. Interactive GenJam: Integrating Real-Time Performance with a Genetic Algorithm, Proc. ICMC 98, Ann Arbor, Michigan, 1998.

Jan Borchers, Designing Interactive Musical Systems: a Pattern Approach, HCI International '99. 8th International Conference on Human-Computer Interaction, Munich, Germany, from 22-27 August, 1999.

Conklin, D. and Witten, Ian H. Multiple Viewpoint Systems for Music Prediction, JNMR, 24:1, 51-73, 1995.

Cope, David. Experiments in Musical Intelligence. Madison, WI: A-R Editions, 1996.

Heuser, Jorg, Pat Martino – His contributions and influence to the history of modern Jazz guitar. Ph.D thesis, University of Mainz (Ge), 1994.

Hiller, L. and Isaacson, A., *Experimental Music*, New York: McGraw-Hill, 1959.

Karma music workstation, Basic guide. Korg Inc. http://www.korg.com/downloads/pdf/KARMA_BG.pdf, 2001.

Lartillot O., Dubnov S., Assayag G., Bejerano G., Automatic modeling of musical style Proc. ICMC 2001, La Habana, Cuba.

New Interfaces for Musical Expression (NIME'01), http://www.csl.sony.co.jp/person/poup/research/chi2000wshp/, 2000.

In proc. of *International Computer music Conference*, Gotheborg (Sweden), ICMA, September 2002.

Orlarey, Y. Lequay, H. MidiShare: a Real Time multi-tasks software module for Midi applications Proc. of ICMC, pp. 234-237, 1989.

Pachet, F. Roy, P. "Automatic Harmonization: a Survey", Constraints Journal, Kluwer, 6:1, 2001.

Ramalho G., Ganascia J.-G. Simulating Creativity in Jazz Performance. Proc. of the AAAI-94, pp. 108-113, Seattle, AAAI Press, 1994.

Ron, D. Singer, Y and Tishby, N. (1996) "The power of amnesia: learning probabilistic automata with variable memory length", Machine Learning 25(2-3):117-149

Triviño-Rodriguez, J. L.; Morales-Bueno, R.; Using Multiattribute Prediction Suffix Graphs to Predict and Generate Music, *CMJ* 25 (3) pp. 62-79 , 2001.

William F. Walker A Computer Participant in Musical Improvisation, Proc. of CHI 1997. Atlanta, ACM Press, 1997.