ImproteK: introducing scenarios into human-computer music improvisation

Jérôme Nika, Marc Chemillier, Gérard Assayag

ImproteK: Introducing Scenarios into Human-Computer Music Improvisation

JÉRÔME NIKA, IRCAM STMS Lab (CNRS, UPMC, Sorbonne Universités)
MARC CHEMILLIER, Cams, Ecole des Hautes Etudes en Sciences Sociales
GÉRARD ASSAYAG, IRCAM STMS Lab (CNRS, UPMC, Sorbonne Universités)

This article focuses on the introduction of control, authoring, and composition in human-computer music improvisation through the description of a guided music generation model and a reactive architecture, both implemented in the software ImproteK. This interactive music system is used with expert improvisers in work sessions and performances of idiomatic and pulsed music and more broadly in situations of structured or composed improvisation. The article deals with the integration of temporal specifications in the music generation process by means of a fixed or dynamic "scenario" and addresses the issue of the dialectic between reactivity and planning in interactive music improvisation. It covers the different levels involved in machine improvisation: the integration of anticipation relative to a predefined structure in a guided generation process at a symbolic level, an architecture combining this anticipation with reactivity using mixed static/dynamic scheduling techniques, and an audio rendering module performing live re-injection of captured material in synchrony with a non-metronomic beat. Finally, it sketches a framework to compose improvisation sessions at the scenario level, extending the initial musical scope of the system. All of these points are illustrated by videos of performances or work sessions with musicians.

Key Words and Phrases: Computer music, computational creativity, interactive music system, live performance, human-computer music improvisation, composed improvisation, scenario, generative model, pattern matching, prefix matching algorithm, mixed offline/online scheduling, discrete reactive system, beat synchronization

This work was supported by the French National Research Agency project DYCI2 ANR-14-CE2 4-0002-01.
Authors' addresses: J. Nika, IRCAM STMS Lab (CNRS, UPMC, Sorbonne Universités), Music Representations team, 1 place Igor Stravinsky, 75004 Paris, France; email: jerome.nika@ircam.fr; M. Chemillier, EHESS, 190 avenue de France, 75244 Paris Cedex 13, France; email: chemillier@ehess.fr; G. Assayag, IRCAM STMS Lab (CNRS, UPMC, Sorbonne Universités), Music Representations team, 1 place Igor Stravinsky, 75004 Paris, France; email: gerard.assayag@ircam.fr.

1. INTRODUCTION

Jazz, blues, or rock improvisation generally relies on a “chord progression” that defines a harmonic structure; the baroque basso continuo leaves it to the performer to realize the harmony by improvising with the right hand from the bass line written for the left hand; improvisation is guided on different levels in the Indian raga (chaining of delimited parts with different lengths and specific identities at the higher level, evolution within each of these parts built from detailed descriptions in terms of melody, tempo, or register). Such a formalized and temporally structured object—not necessarily described with a harmonic vocabulary—can be found in various repertoires of improvised performance.

Outside the musical scope, this notion can be transposed to commedia dell’arte, where the “canovaccio” outlines the sequence of events and situations of the plot that the representation will follow, or to the oral tradition of folktales or epic tales for which the improvised narration is based on an outline made of elementary units, sometimes associated to a set of established formulas defining a grammar of the narration.

The idea of a preexisting temporal structure guiding the improvised performance is considered in the cognitive studies dealing with improvisation that address the issue of planning associated with the need for reactivity characterizing improvisation. This entity is broader than the formal scenario because it covers all the narrative strategies of an improviser. For example, the notion of “plan” for Sloboda [1982] or Shaffer [1980] is defined as an abstract object symbolizing the fundamental structure of the performance, letting the finer dimensions to be generated or organized in due time. Beyond the mere sequence of formalized constraints, Pressing [1984] introduces the “referent” as an underlying scheme guiding or aiding the production of the musical material. The referent extends to cognitive, perceptive, or emotional dimensions, and its relation with the improvised behavior can be, among others, metaphoric, imitative, allegoric, or antagonistic.

Human-computer improvisation systems generate music on the fly from a model and external inputs, typically the output of an human musician’s live improvisation. In this article, we present some music generation models and a scheduling architecture implemented in the music improvisation system ImproteK, used in concerts by renowned improvisers.[1] This work introduces authoring and control in this process and combine the ability to react to dynamic controls with that of maintaining conformity to a fixed or dynamic temporal specification. The improvisation model presented here relies on a “scenario” structure. As in the examples of improvisation styles previously mentioned, this object does not carry the narrative dimension of the improvisation, that is, its fundamentally aesthetic and non-explicit evolution, but is a sequence of formalized constraints for the machine improvisation.

After presenting different approaches of guided improvisation and giving some background in Section 2, Section 3 introduces the general principle and the motivation of the “scenario/memory” music generation model. Section 4 and Section 5 go deeper into scientific and technical details.

Section 4 details the algorithms involved in the “scenario/memory” generation model. They ensure the conformity of the machine improvisation to the specifications provided by the scenario and introduce anticipation in the guided music generation process.

Section 5 describes how this generation model is used during a performance. It shows how anticipation is combined with reactivity by embedding this offline model

[1]Compilation of short video excerpts: **Video 0:** www.youtube.com/watch?v=OhXf-1QpqEI. The links to the complete videos are given in the following sections. See the descriptions of the videos in the appendix at the end of the journal.

into a reactive framework using a mixed offline/online architecture. This section also describes the articulation of this reactive architecture with an audio rendering module that enables the system to re-inject live audio material to generate improvisations matching the scenario and synchronized with a non-metronomic pulse to adapt to the environment.

Finally, Section 6 underlines the genericity of the scenario/memory approach and widens to the issue of introducing authoring and control in human-computer music improvisation by sketching a framework to compose improvisation sessions at the scenario level.

2. RELATED WORKS AND BACKGROUND

We focus here on interactive music systems integrating a notion of *guidance*. A number of existing improvisation systems drive the music generation processes by involving a user steering their parameters. First, this user control can concern (low-level) system-specific parameters. This is, for example, the case with Omax [Assayag et al. 2006b; Lévy et al. 2012], which is controlled by an operator-musician steering the navigation through a representation extracted in real time from the playing of a live musician, or Mimi4x [François et al. 2013], which involves a user in the construction of the performance by choosing the musical corpus and modifying the generation parameters.

We refer here to *guided improvisation* when the control on music generation follows a more declarative approach, that is, specifying targeted outputs or behaviors using an aesthetic, musical, or audio vocabulary independent of the system implementation, whether this control is short term or long term (see Nika [2016] for a comprehensive review).

On the one hand, *guiding* is seen as a purely reactive and step-by-step process. SoMax [Bonnasse-Gahot 2014], for instance, translates the musical stream coming from an improviser into activations of specific zones of the musical memory in regards to a chosen dimension (for example, the harmonic background). VirtualBand [Moreira et al. 2013], emphasizing interaction and reactivity, also extracts multimodal observations from the musician's playing to retrieve the most appropriate musical segment in the memory in accordance to previously learned associations. In the same line, Reflexive Looper [Pachet et al. 2013] uses in addition some harmonic annotations in the research criteria. It should be noticed that some systems do not use concatenative synthesis of live or prerecorded musical material but generate improvisations from predefined generative models with which a musician interacts during the performance. Among them, Sioros and Guedes [2011a, 2011b] use a rhythmic analysis of the live inputs to steer generative models with a focus on syncopation.

On the other hand, *guiding* means defining upstream temporal structures or descriptions driving the generation process of a whole music sequence. Pachet and Roy [2011], for instance, use constraints in such a music generation process. ReChord [Ramona et al. 2015] is an offline engine based on chord progressions generating new accompaniment tracks from a single recording of accompaniment, using concatenative synthesis at the chord scale. More broadly, the general approach of the Flow Machines project [Ghedini et al. 2016] is to apply a *style* (corpus) to a well-chosen *structure* (sequential content such as text or music) to generate creative objects. In the works of Donzé et al. [2014], the concept of "control improvisation" [Fremont et al. 2014] applied to music also introduces a guiding structure via a reference sequence and a number of other specifications. This structure is conceptually close to the scenario used in our approach. PyOracle [Surges and Dubnov 2013] proposes to create behaviour rules or scripts for controlling the generation parameters of an improvisation generation using "hot spots" (single event targets). Wang and Dubnov [2014] extend this work in an offline architecture using sequences instead of single events as query targets. This idea of mid-term temporal queries shares common issues with the playing mode involving a dynamic scenario presented in this article.

The purely reactive approach offers rich interaction possibilities but does not integrate prior knowledge about the temporal evolution. On the other hand, steering music generation with mid- or long-term structures enables anticipation but lacks reactivity with regard to external or user controls. This article devises an architecture at an intermediate level between the reactive and offline approaches to combine anticipations relative to a predefined plan and dynamic controls. The proposed architecture is structured around an offline generation module based on a scenario, embedded in a reactive framework steered by external events. The generation module is called to produce short-term anticipations matching the scenario and may eventually rewrite these anticipations over time according to incoming control events.

ImproteK, presented here, and Omax, Somax, and PyOracle belong to a family of related researches and implementations on machine improvisation carried out at Ircam (Institut de recherche et coordination acoustique/musique), UCSD (University of California, San Diego), and EHESS (École des Hautes Études en Sciences Sociales). They share a sequential model (that we call here "memory") learned from live or offline music streams that is explored interactively at performance time with various types and degrees of constraints. Indeed, the generation model follows the work on statistical style modeling initiated in Assayag et al. [1999] and Dubnov et al. [1998] and its implementation in the real-time improvisation system Omax [Assayag et al. 2006a, 2006b; Lévy et al. 2012]. A "beat," long-term constraints, and *a priori* knowledge were then introduced into the generation process with the first version of ImproteK [Nika and Chemillier 2012] by means of a formalism conveying different musical notions depending on the applications, like meter as regards rhythm or chord notation as regards harmony, in the line of works on the use of chord progressions in improvisation [Chemillier 2001, 2004, 2009]. The music generation model described in this article generalizes these long-term constraints with the "scenario."

The real-time architecture embedding the generation model is at the frontier between the offline and online paradigmatic approaches in computer music systems regarding time management and planning or scheduling strategies. This frontier is studied in current works in computer music such as that by Agostini and Ghisi [2013], Echeveste et al. [2013a], Bresson and Giavitto [2014], and Bouche and Bresson [2015]. On the one hand, "offline" corresponds to computer-assisted composition systems [Assayag 1998] where musical structures are computed following best effort strategies and where rendering involves static timed plans. On the other hand, "online" corresponds to performance-oriented systems [Dannenberg 1989] where the computation time is part of the rendering.

3. A SCENARIO TO DEAL WITH CONFORMITY, ANTICIPATION, AND HYBRIDIZATION

The system described in this article is initially dedicated to the scope of "idiomatic" music improvisation [Bailey 1993]. The improvisation process in such a formalized context is modeled as re-injections, transformations, and recontextualizations of elements that have been eared or played in a different context defined with the same vocabulary, by analogy with a musician re-injecting a "cliche" when improvising over a given local harmonic progression. In this view, musicality lies in the anticipated or unexpected nature of these re-injections. This section introduces the musical motivations and the principle of the scenario/memory generation model, which will be detailed in Section 4.

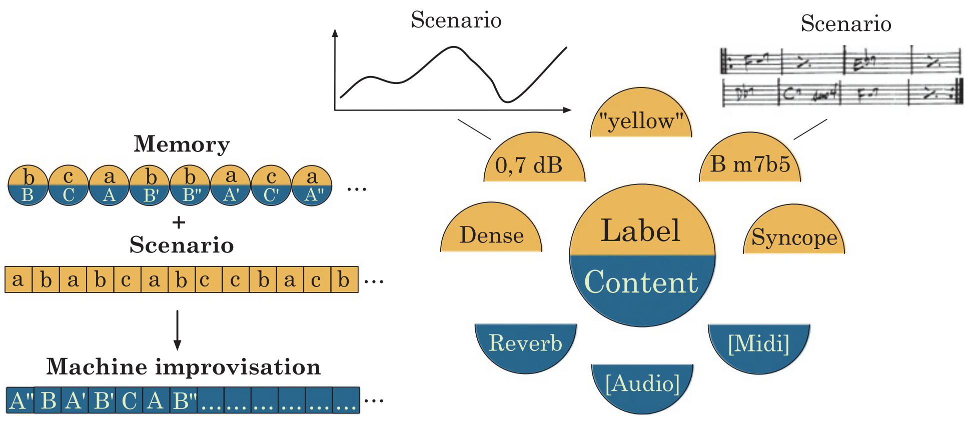

Fig. 1. An *event*: the elementary unit of the musical memory. It is constituted by a musical *content* annotated by a *label*. The scenario guides the concatenation of these contents to generate the machine improvisation.

3.1. "Scenario" and "Memory"

The generation model articulates a *scenario* guiding the generation and a structured and indexed *memory* in which musical sequences or events are searched and retrieved to be transformed, rearranged, and reordered to create new improvisations:

—the *scenario* is a symbolic sequence guiding the improvisation and defined on an appropriate alphabet (depending on the musical context),
—the *memory* is a sequence of contents labeled with a symbolic sequence defined over this same alphabet.

The scenario can be any sequence defined over a chosen alphabet suitable for the musical context, for example, a harmonic progression in the case of jazz improvisation or a discrete profile describing the evolution of audio descriptors to control concatenative audio sound synthesis.

The musical memory used during an improvisation session is a store of continuous musical sequences (e.g., MIDI[2], audio, parameters for sound synthesis) represented as sequences of events. An *event* (Figure 1) has a duration and is indexed by its *position* (index $\in \mathbb{N}$), which will also be called *date* (both are equivalent since the time from the beginning of the sequence is stored within each event). The memory can be constituted online by recording the music played by the human co-improvisers during a live performance (the way musical inputs from the musicians are segmented into events, annotated, and learned in real time is described in Section 5.1) and/or offline (from annotated material). All the sequences in the memory have to be segmented into events annotated using the alphabet chosen for the scenario (e.g., harmonic labels) but do not have to be created within the same source scenario (e.g., a set of recordings of solos on different jazz standards, see Section 3.3).

In this context, "improvising" means navigating through the memory in a creative way to collect some contiguous or disconnected sequences matching the successive parts of the scenario and concatenating them to create a musical phrase. We will first consider that an event in the memory *matches* a label of the scenario when the labels are equals. Yet, the events in the memory can also be transformed to virtually increase the size of the memory. The generic approach (equivalence classes on the labels associated with transformations of the contents) will be developed in Section 6, and a first example will

[2]Musical Instrument Digital Interface, standard music technology protocol: https://www.midi.org.

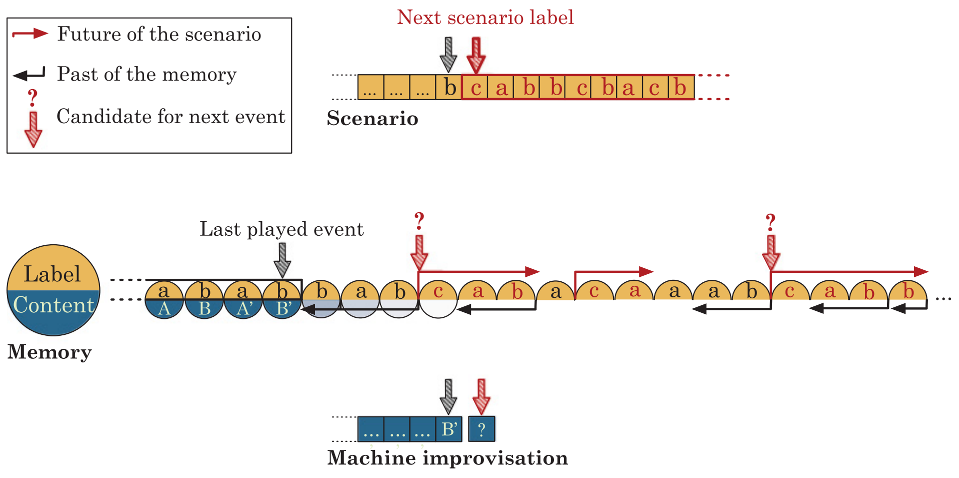

Fig. 2. Using the scenario to introduce anticipation in the music generation process.

be given in Section 3.3 with the case of transposition when the scenario is defined as a harmonic progression.

3.2. Music Generation Guided by a Scenario: Combining Anticipation and Digression

Conformity. In first approach, the scenario ensures the conformity of the machine improvisation regarding the stylistic norms and aesthetic values implicitly carried by the musical idioms. **Video 1** gives a first example of how the scenario/memory model recontextualizes some subsequences of the memory to generate new musical sequences matching the scenario.[3] In this example of jazz co-improvisation, the solo played by the musician is segmented in real time using beat markers and annotated with harmonic labels so it can be immediately re-injected by the model to produce an improvisation matching the scenario that is here a simple chord progression.

Anticipation. In the second approach, the scenario gives access to prior knowledge of the temporal structure of the improvisation that is exploited to introduce *anticipation* in the generation process, that is, to take into account the future of the scenario to generate the current time of the improvisation. This point is particularly relevant in the case of "hybrid" improvisation: when the scenario and the memory are different sequences defined with the same alphabet (see an illustration with the example of jazz improvisation in Section 3.3).

Digression. The scenario/memory model searches for the continuity of the musical discourse by exploiting the similar patterns in the sequences and the ability to create new content that goes beyond copy using the regularities in the memory (see the algorithms in Section 4). This last condition maintains the coherence of the musical discourse when digressing from the original material, that is, when non-contiguous subsequences of the memory are chained in the machine improvisation. Figure 2 represents a step in the improvisation process. It combines anticipation and coherence with the musical logic of the memory by searching for events ensuring both *continuity with the future of the scenario* (red arrows) and *continuity with the past of the memory* (black arrows).

[3]**Video 1:** www.youtube.com/watch?v=w17pFvrI06A. See the description of the associated artistic collaboration in the online appendix section at the end of the journal.

Finally, we will show in Section 6 that the scenario approach can be used to go beyond simple conformity criteria and widens to the *composition of improvised performance* in an idiomatic context or not.

3.3. Example of Jazz Improvisation

The example of jazz improvisation illustrates how anticipation can be used to create "hybrid" improvisations and how a transformation such as transposition can be used to virtually increase the size of the memory. In the case of a scenario defined as the chord progression of a jazz standard and a memory recorded on different chord progressions, the basic idea is the following: If the scenario requires a *ii-V-I* progression, retrieving an instance of *ii* located in a *ii-V-I* progression, then an instance of *V* located in a *V-I* progression is likely to produce a better result than the concatenation of an instance of *ii*, an instance of *V*, and an instance of *I* independently retrieved in the memory.

Depending on the nature of the alphabet, adapted heuristics can be defined to complete the generic algorithm. In this example, we define the scenario and the memory on the alphabet constituted by the four notes chords deriving from the harmonization of the major scale commonly found in jazz harmony. Figure 3 presents the possible evolutions through the jazz standard "Blue in Green" (scenario) only using the longest factors retrieved from the harmonic progression of "Autumn Leaves" (memory).

Here we define equivalence on the labels modulo transposition and the associated transformation of the retrieved contents (see Section 6.1 for the generic approach). When using a harmonic alphabet, more complex equivalences can be defined, for example, transformations defined by chord substitution grammars [Chemillier 2004]. To simplify the example in Figure 3, only the longest factors are represented: An arrow leaving a state of the scenario points on the furthest state that can be reached following a sequence extracted from the memory in its original or transposed state. The number of sequences matching this longest path is given for each of the relevant transpositions. In the case of jazz improvisation, transposition is an example of control on music generation: Depending on the musical situation, one can sometimes prefer the longest paths whatever necessary transposition jumps (which may introduce discontinuities) and sometimes choose the paths minimizing the transpositions even if some progressions or complete cadences that could be present in the memory with a different local tonality may be dismissed.

Video 2a shows an example of such hybrid improvisations during a concert using an early MIDI version of the system.[4] The scenario is defined as a harmonic progression and the memory is a heterogeneous sets of different jazz standards and ballads coming from previous improvisation sessions with different musicians (in particular Bernard Lubat and Jovino Santos Neto).

With an other approach, **Video 2b** shows the finale of an improvisation by pianist Hervé Sellin playing with the system.[5] Its musical memory contains different recordings by Billie Holiday, Edith Piaf, and Elisabeth Schwartzkopf (singing Puccini, Mozart, and Mahler). The aim was here to create a quatuor with a live musician and a "virtual trio" with a patchwork aesthetics.

4. SCENARIO/MEMORY MUSIC GENERATION ALGORITHMS

This section presents the algorithms involved in the scenario/memory generation model introduced in the previous section. Section 4.1 outlines the general algorithm,

[4]**Video 2a:** www.youtube.com/watch?v=yY3B5qfFri8. See the description of the associated artistic collaboration in the online appendix section at the end of the journal.
[5]**Video 2b:** www.youtube.com/watch?v=reJ-SiblCcs. See the description of the associated artistic collaboration in the online appendix section at the end of the journal.

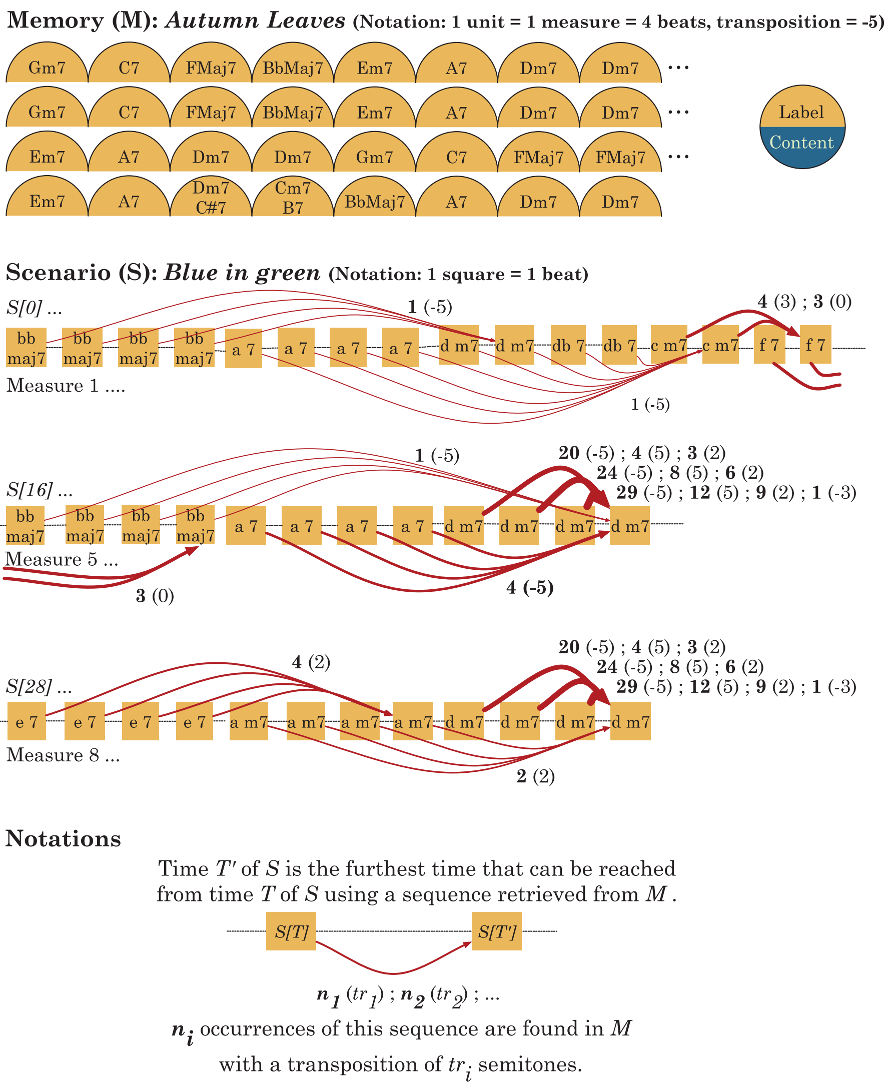
Memory (M): Autumn Leaves (Notation: 1 unit = 1 measure = 4 beats, transposition = -5)
Gm7 C7 FMaj7 BbMaj7 Em7 A7 Dm7 Dm7 ...
Gm7 C7 FMaj7 BbMaj7 Em7 A7 Dm7 Dm7 ...
Em7 A7 Dm7 Dm7 Gm7 C7 FMaj7 FMaj7 ...
Em7 A7 Dm7 C#7 Cm7 B7 BbMaj7 A7 Dm7 Dm7
Label
Content
Scenario (S): Blue in green (Notation: 1 square = 1 beat)
S[0] ...
1 (-5)
4 (3) ; 3 (0)
bb maj7 bb maj7 bb maj7 bb maj7 a 7 a 7 a 7 a 7 d m7 d m7 db 7 db 7 c m7 c m7 f 7 f 7
Measure 1 ....
1 (-5)
S[16] ...
1 (-5)
20 (-5) ; 4 (5) ; 3 (2)
24 (-5) ; 8 (5) ; 6 (2)
29 (-5) ; 12 (5) ; 9 (2) ; 1 (-3)
bb maj7 bb maj7 bb maj7 bb maj7 a 7 a 7 a 7 a 7 d m7 d m7 d m7 d m7
Measure 5 ...
3 (0)
4 (-5)
S[28] ...
4 (2)
20 (-5) ; 4 (5) ; 3 (2)
24 (-5) ; 8 (5) ; 6 (2)
29 (-5) ; 12 (5) ; 9 (2) ; 1 (-3)
e 7 e 7 e 7 e 7 a m7 a m7 a m7 a m7 d m7 d m7 d m7 d m7
Measure 8 ...
2 (2)
Notations
Time T′ of S is the furthest time that can be reached from time T of S using a sequence retrieved from M.
S[T]
S[T′]
n1 (tr1) ; n2 (tr2) ; ...
ni occurrences of this sequence are found in M with a transposition of tri semitones.

Fig. 3. Example using a harmonic alphabet: covering "Blue in Green" using sequences in an interpretation of "Autumn Leaves" (simplified representation: only the longest sequences).

Section 4.2 focuses on the prefix indexing algorithm handling the continuity with the future of the scenario, and Section 4.3 explains how the continuity with the past of the memory is obtained from the automaton structure chosen to learn the musical material.

4.1. Overview of the Guided Music Generation Algorithm

4.1.1. Notations and Definitions. The scenario and the sequence of labels describing the musical memory are represented as words on an alphabet A. Choosing an alphabet A for the labels of the scenario and the memory sets the equivalence classes labeling the musical contents in the memory (see Section 6 for some examples of alphabets).

Given a scenario S of length s, the letter at index T in S is denoted by $S[T]$. After defining a temporal unit for the segmentation, $S[T]$ is the required label for the time T

of the improvisation. Given a memory M of length m, the letter at index k in M is denoted by $M[k]$. $M[k]$ is the equivalence class labeling the musical event corresponding to the date k in the memory M. In the following descriptions the memory will be assimilated to the word M. The labels and contents in the the memory will be distinguished when necessary using lowercase letters and uppercase letters, respectively. For example, different musical contents B', B'', B''', ... belonging to a same equivalence class b will be labeled by b.

Finally, the machine improvisation, that is, the sequence of indexes of the events retrieved in M and concatenated to generate the improvisation, will be denoted by $\{i_T\}_{0 \leq T < s}$.

Using the usual vocabulary, the zero letter sequence is called the *empty string* and is denoted by ϵ. A string x is a *factor* of a string y if there exist two strings u and v such that $y = uxv$. When $u = \epsilon$, x is a *prefix* of y; and when $v = \epsilon$, x is a *suffix* of y.

4.1.2. Current Scenario at Date T. The scenario gives access to a prior knowledge of the temporal structure of the improvisation to play. Anticipation can therefore be introduced by taking into account the required labels for the future dates to generate improvisation at current time T. The *current scenario* at date T, denoted by S_T, corresponds to the suffix of the original scenario beginning at the letter at index T: $S_T = S[T] \ldots S[s-1]$. At each time T, the improvisation goes on from the last state i_{T-1} retrieved in the memory, searching to match the current suffix S_T of the scenario.

4.1.3. Anticipation and Digression: Definition of Index Sets. As introduced in Section 3.2, the model combines anticipation by ensuring continuity with the future of the current scenario S_T and coherence with the musical logic of the memory M when digressing by maintaining continuity with the past of the memory. To achieve this, we define the following index sets of the memory M that are used in the scenario/memory generation algorithm (Section 4.1.4):

$$\begin{aligned}\underline{Future_S(T)} &= \text{Positions in } M \text{ sharing a common future with the current scenario } S_T\\ &= \{k \in \mathbb{N} \mid \exists\, c_f \in \mathbb{N},\ M[k] \ldots M[k + c_f - 1] \in \text{ Prefixes}(S_T)\},\ T \in [0, s[.\end{aligned}$$

$k \in Future_S(T)$ is the left position of a factor of M equal to a prefix of the current scenario S_T. $M[k]$ shares a common future with $S[T]$ and provides *continuity with the future of the scenario* measured by the length c_f of the prefix. The maximum length c_f associated to an index $k \in Future_S(T)$ is denoted by $c_f(k, T)$.[6]

$$\begin{aligned}\underline{Past_M(i)} &= \text{Positions in } M \text{ sharing a common past with the event } M[i]\\ &= \{k \in \mathbb{N} \mid \exists\, c_p \in [1, k],\ M[k - c_p + 1] \ldots M[k] \in \text{Suffixes}(M[0] \ldots M[i])\},\ i \in [0, m[.\end{aligned}$$

$k \in Past_M(i)$ is the right position of a factor of M equal to a suffix of $M[0] \ldots M[i]$. $M[k]$ shares a common past with $M[i]$ and provides *continuity with the past of the memory* measured by the length c_p of the suffix. The maximum length c_p associated to an index $k \in Past_M(i)$ is denoted by $c_p(k, i)$.[7]

$$\begin{aligned}\underline{Chains_{S,M}(T, i)} &= \text{Positions in } M \text{ starting a sequence chaining with } M[i] \text{ at time } T\\ &= \text{Positions in } M \text{ sharing a common future with the current scenario } S_T\\ &\quad \text{and preceded by a sequence sharing a common past with the event } M[i]\\ &= \{k \in \mathbb{N}^* \mid k \in Future_S(T) \text{ and } k - 1 \in Past_M(i)\},\ T \in [0, s[,\ i \in [0, m[.\end{aligned}$$

[6]See Section 4.2 for the algorithms to build $Future_S(T)$.
[7]See Section 4.3 for the algorithms to build $Past_M(i)$.

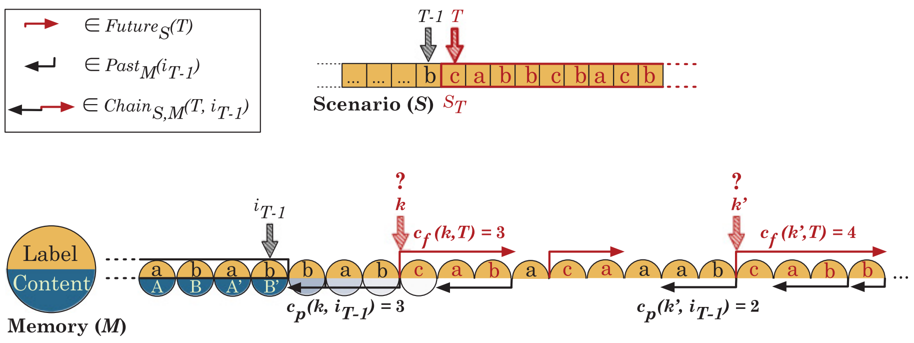

Fig. 4. Construction of $Chains_{S,M}(T, i_{T-1}) = \{k, k'\}$: Positions in M sharing a common future with the current scenario S_T and preceded by a sequence sharing a common past with the event $M[i_{T-1}]$.

As illustrated in Figure 4, $k \in Chains_{S,M}(T, i)$ shares a common future with the current scenario S_T and is preceded by a sequence sharing a common past with the event at index i. The indexes k in $Chains_{S,M}(T, i)$ are the left positions of sequences of length $c_f(k, T)$ constituting possible fragments of improvisation starting at time T. Besides, they offer smooth transitions from $M[i]$ thanks to their common past of length $c_p(k, i)$, assuming that jumps between two segments sharing a common past preserve a certain musical homogeneity.

The sequences starting at positions $k \in Chains_{S,M}(T, i)$ can be simply copied.[8] To go beyond simple copy, $k \in Chains_{S,M}(T, i)$ can be used as a starting point to follow an equivalent non-linear path using regularities of the memory to jump to other zones sharing a common past with the previously retrieved slice. To cover both cases, we define the set of possible *continuations* from the index $i \in [0, m[$ in M at date $T \in [0, s[$ as

$$\begin{aligned}\underline{Conts_{S,M}(T, i)} &= \text{Possible continuations from } M[i] \text{ at time } T\\ &= \text{Positions in } M \text{ matching the label } S[T] \text{ of the scenario and}\\ &\quad \text{preceded by a sequence sharing a common past with the event } M[i]\\ &= \{k \in \mathbb{N}^* \mid M[k] = S[T] \text{ and } k-1 \in Past_M(i)\}.\end{aligned}$$

Finally, i_T is chosen in $Conts_{S,M}(T, i_{T-1})$.

4.1.4. Guided Generation Algorithm. The generation process is divided in *phases* constrained by suffixes of the scenario.[9] Figure 5 gives an example of two consecutive generation phases. Each phase consists in two successive steps involving the previously defined index sets.

1-Anticipation: Find an event in the memory sharing a common future with the scenario while ensuring continuity with the past of the memory. *2-Copy or digression*: Retrieve the whole sequence (example of the phase S_T, black) or use the regularities in the memory to follow an equivalent non-linear path (example of the phase $S_{T'}$, red).

[8]In this case, given $k \in Chains_{S,M}(T, i)$, $\forall l \in [0, c_f(k, T)[$, $i_{T+l} = k + l$.

[9]These navigation phases through the memory segment the algorithmic process of producing the improvisation, but they do not correspond in general to distinct musical "phrases."

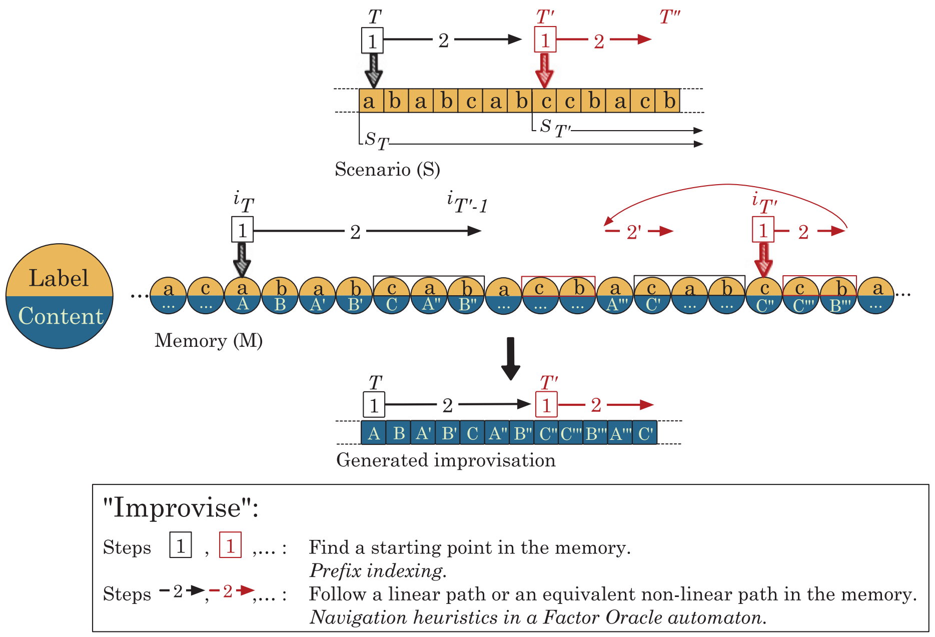
T
T'
T''
a b a b c a b c c b a c b
S_T
$S_{T'}$
Scenario (S)
i_T
$i_{T'-1}$
$i_{T'}$
2'
Label
Content
Memory (M)
Generated improvisation
A B A' B' C A'' B'' C'' C''' B''' A''' C'
"Improvise":
Steps 1, 1, ... : Find a starting point in the memory.
Prefix indexing.
Steps 2, 2, ... : Follow a linear path or an equivalent non-linear path in the memory.
Navigation heuristics in a Factor Oracle automaton.

Fig. 5. Scenario/memory generation model: Example of two successive generation phases, S_T (black) and then $S_{T'}$ (red). The first one generates a fragment of improvisation starting from date T satisfying the current scenario S_T. At the end of this first phase, the prefix $S[T] \ldots S[T'-1]$ of S_T has been processed. A new research phase over the suffix $S_{T'}=S[T'] \ldots S[s-1]$ of S is then launched to complete the improvisation up to $T''-1$. In the example of this second phase (red), the regularities in the memory indexed by the sets $Past_M(i)$ are used to follow a non-linear path..

Formally, the generation algorithm is summarized in Algorithm 1 and consists in

(1) *Anticipation: searching for a starting point* (lines 3–8 in Algorithm 1, steps 1 in Figure 5).
The search first looks for events in M sharing a common future with the current scenario S_T and a common past with the last retrieved index i_{T-1} in M, that is, $Chains_{S,M}(T, i_{T-1})$. When none of the events in M can provide both continuity with the future of the scenario and continuity with the past of the memory, only the first criterion is searched, that is, the continuity with the future of the current scenario. If no solution is found, then alphabet-dependent transformations are used (see Section 6).
(2) *Copy or digression: Navigating through the memory* (lines 9–11 in Algorithm 1, steps 2 in Figure 5). After finding a factor of M matching a prefix of S_T, it can be copied or used as a starting point to follow an equivalent non-linear path in the memory using the continuations in $Conts_{S,M}(T, i_{T-1})$ until launching a new phase is necessary.

At each of theses steps, the concerned index sets are built and the selection among the candidate positions is done in order to satisfy a set of *secondary generation parameters* containing all the parameters driving the generation process that are independent from the scenario: parametrization of the generation model and content-based constraints to filter the set of possible results returned by the algorithm (see 5.1).

ALGORITHM 1: Guided Generation Algorithm

```
Input: Scenario $S$ of length $s$, Memory $M$ of length $m$.
Output: $\{i_T\}_{0 \le T < s}$, indexes of the slices retrieved in $M$ and concatenated to generate the
        improvisation.
1  $T = 0$;
2  while $T < s$ do
3                                                        /* Step (1): Starting point */
4      if $T > 0$ and $Chain_{S,M}(T, i_{T-1}) \neq \emptyset$ then
5          $i_T \leftarrow k \in Chain_{S,M}(T, i_{T-1})$;
6      else
7          $i_T \leftarrow k \in Future_S(T)$ or alphabet-specific heuristics.
8      end
9      $T$++;
10                                                           /* Step (2): Navigation */
11     while $T < s$ and $Cont_{S,M}(T, i_{T-1}) \neq \emptyset$ do
12         $i_T \leftarrow k \in Cont_{S,M}(T, i_{T-1})$;
13         $T$++;
14     end
15 end
```

4.2. Continuity with the Future of the Scenario

$Future_S(T)$ (defined in Section 4.1.3) is the index set of M defined to deal with continuity with the future of the scenario. This section gives an overview of the proposed algorithm to build $Future_S(T)$ and then $Chain_{S,M}(T, i_{T-1})$. The construction of $Future_S(T)$ comes down to the general problem of indexing the prefixes of a pattern $X{=}X[0]\dots X[x-1]$ in a word $Y{=}Y[0]\dots Y[y-1]$.

4.2.1. Outline of the Algorithm. As illustrated in Figure 6, the algorithm for indexing the *prefixes* of X in Y follows the outline of the classic algorithms for indexing the *occurrences* of a pattern in a word: comparisons and calls to a failure function to shift a sliding comparison window in such a way that redundant comparisons are avoided. The algorithm presented below uses the failure function f of the Morris-Pratt algorithm [Morris and Pratt 1970] that indexes the occurrences of the pattern X in Y by describing the run of the deterministic finite automaton recognizing the language A^*X (where A is the alphabet on which X and Y are defined) on the word Y. The algorithm for indexing the prefixes of X in Y is divided into a preprocessing phase on the pattern X and a searching phase represented in Figure 6. The preprocessing phase provides the tools used in the searching phase: the failure function f and the function B defined below.

4.2.2. Preprocessing Phase. A *proper factor* of X is a factor of X that differs from X and a *border* of a non-empty string X is a proper factor of X that is both a prefix and a suffix of X. We define f as follows (see Crochemore et al. [2007] for the construction of the borders table and f):

$$\forall i \in [1, x],\ f(i) = \text{length of the longest border of } X[0]\dots X[i-1],\ \ f(0) = -1.$$

f is used as a *failure function* in the algorithm: f computed on the pattern X indexes some regularities in X that are used in the searching phase to shift the sliding window from the last index i to a relevant index $f^k(i)$ so unnecessary comparisons are avoided (last step in Figure 6).

f is defined in such a way that when a prefix is found during the searching phase, it is the longest prefix of X at the concerned position of Y, and the indexes it covers will not be visited during the following attempts. A shift of the sliding window has to be *valid*, that is, it has to ensure that no prefixes are forgotten when sliding the window.

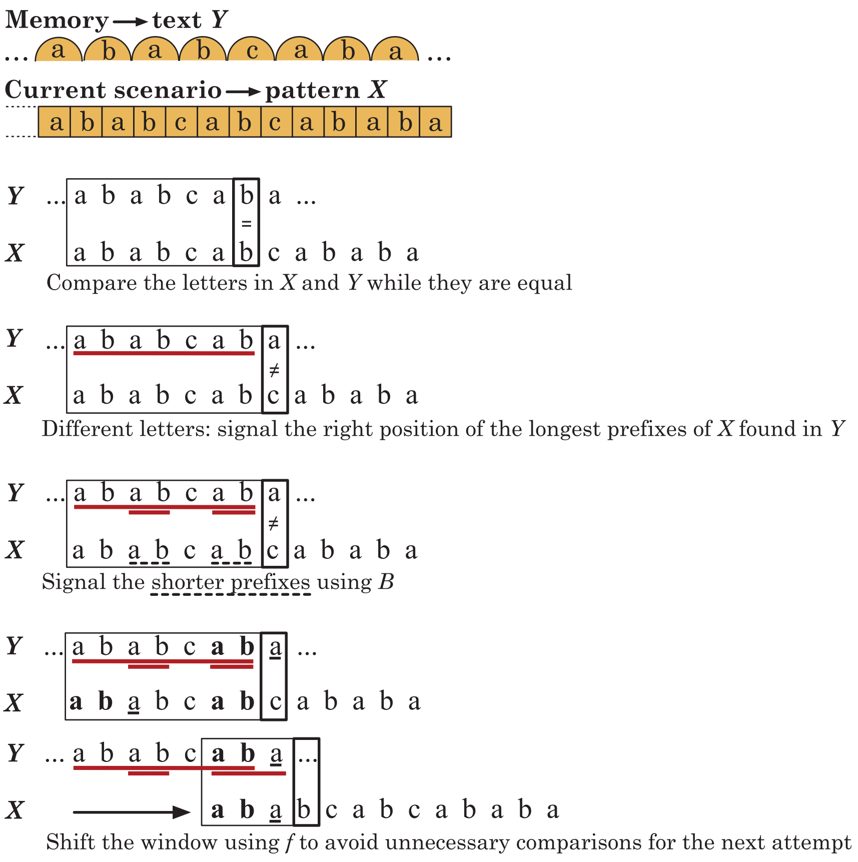
Memory → text Y
... a b a b c a b a ...
Current scenario → pattern X
a b a b c a b c a b a b a
Y ... a b a b c a b a ...
X a b a b c a b c a b a b a
Compare the letters in X and Y while they are equal
Different letters: signal the right position of the longest prefixes of X found in Y
Signal the shorter prefixes using B
Shift the window using f to avoid unnecessary comparisons for the next attempt

Fig. 6. Indexing the prefixes of a pattern X in a text Y.

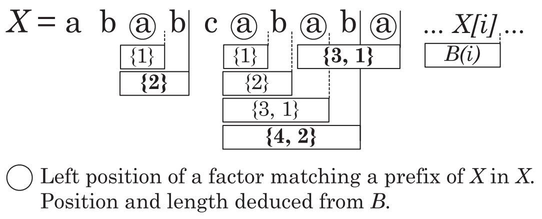
X = a b a b c a b a b a ... X[i] ...
{1}
{2}
{1}
{2}
{3, 1}
{4, 2}
{3, 1}
B(i)
Left position of a factor matching a prefix of X in X.
Position and length deduced from B.

Fig. 7. $B(i)$: Sets of the lengths of the borders of $X[0]\dots X[i]$. The locations of the non-trivial occurrences of all the prefixes of the pattern X in X itself are then deduced from B (rectangles).

We have therefore to signal the shorter prefixes that may be included in the previously found longest prefix before sliding the window (third step in Figure 6). This consists in reporting the relevant prefixes of X within X itself in this found prefix. In addition to its use as a failure function, we therefore use f to obtain the locations of the prefixes of X within X itself. To do so, we define the function B, with $B(i)$ the set of the lengths of all the borders of $X[0]\dots X[i]$:

$$\forall i \in [1, x-1],\ B(i) = \{\text{length of } b \mid b \text{ border of } X[0]\dots X[i]\}.$$

As illustrated in Figure 7, the locations of prefixes of X within X itself and their lengths are immediately deduced from B (rectangles in Figure 7), and B is directly obtained from f: Indeed, by a simple recurrence (see Crochemore et al. [2007]):

$$\forall i \in]1, x[,\ B(i-1) = \{f(i),\ f^2(i), \dots,\ f^n(i) > 0\}.$$

4.2.3. Searching Phase: Indexing the Prefixes of the Current Scenario S_T in the Memory M. Finally, with $X = S_T$ and $Y = M$, $Future_S(T)$ is built as follows:

(1) Preprocessing phase, construction of f on S_T, hence B.
(2) Searching phase, index the prefixes of S_T in M:
  Comparisons of the letters in S_T and M, when $S_T[i] \neq M[j]$ and $i > 0$:
  (a) $(j-1)$ is the right position of a prefix of length i of S_T found in M:
    $(j-i) \in Future_S(T)$
    This is the longest prefix of S_T in M ending in $j-1$.
  (b) Use B to signal the shorter prefixes of S_T in $M[j-i] \dots M[j-1]$.
  (c) Go backward in S_T with f to avoid unnecessary comparisons for the next attempt.
(3) Combine with $Past_M(i)$ (Section 4.3) to get $Chain_{S,M}(T, i)$.

The positions and the lengths of the prefixes are recorded in a table. The lengths of the prefixes correspond to the parameters c_f introduced in Section 4.1.3.

To summarize, the searching phase indexes the prefixes of S_T in M by signaling some locations dismissed in the Morris-Pratt algorithm, without proceeding to extra comparisons or moves backwards. This simple implementation justifies the choice of the failure function of the Morris-Pratt algorithm. Indeed, although f is not optimized (for example, in comparison to that of Knuth et al. [1977] or Boyer and Moore [1977]), it enables us to move easily from the research of occurrences to the research of prefixes. Furthermore, this algorithm runs in time $\Theta(m)$ and does not exceed $2 * m - 1$ comparisons, and the preprocessing phase runs in time $\Theta(s - T)$. This execution time and this failure function proved to be suitable for our use case (this result is empirical: The improvisation system in which this model is implemented has been used many times during performances and work sessions with expert musicians). Indeed, the first fields of musical experimentation with the model were jazz chord progressions, and processed sequences contained therefore multiple regularities. Because of the harmonic rhythm (generally two or four beats), they are often of form $\dots x^4y^2z^2x^8 \dots$ (more details on the first version of the algorithm in Nika and Chemillier [2014]).

4.3. Continuity with the Past of the Memory

Continuity with the past of the memory is handled with $Past_M(i)$ introduced in Section 4.1.3. The sets $Past_M(i)$ are used both to filter the set of sequences sharing a common future with the current scenario to get the chaining sequences $Chain_{S,M}(T, i)$ (Section 4.1.3) and to add non-linear paths to the set of possible continuations $Cont_{S,M}(T, i)$ when navigating through the memory (Section 4.1.3). This paragraph details how these sets are obtained from the automaton structure chosen to learn the musical memory: the Factor Oracle (FO) automaton [Allauzen et al. 1999; Lefebvre et al. 2002] (Figure 8).

This automaton was introduced to remedy the difficulty to build a deterministic finite automaton recognizing the language constituted by the factors of a word. The FO built on a word X is a deterministic finite automaton accepting at least the factors of X (for each of these words, there exists at least one path labeled by it in the automaton, leading to a final state). In the context of a musical application, this automaton presents the advantage of keeping the sequential aspect of the temporally structured musical memory and does not aggregate in the same state all the contents labeled by the same equivalence class. Moreover, its construction algorithm (see Allauzen et al. [1999]) is

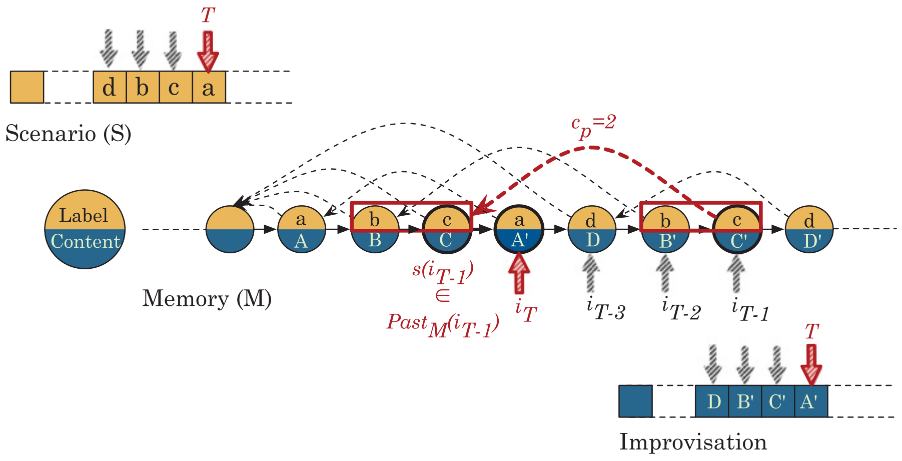

Fig. 8. Using the regularities of the memory (s suffix link function of the FO memory) to follow non-linear paths (continuations) or chain disconnected sequences while preserving musical coherence.

incremental and linear in time in space. It is therefore particularly relevant for real-time applications (see Assayag et al. [2006b] for the application of this automaton to the issue of *stylistic reinjection* for human-computer music improvisation).

Like in most of the systems using the FO for music improvisation, the automaton is not used here to proceed to proper pattern matching but in an indirect way: Some construction links, the *suffix links*, carry the information to build the sets $Past_M(i)$. The FO construction function, the *suffix link function* s, is used to index regularities in the memory. The function s computed on a word X is defined as follows:

$$s(i) = \text{leftmost position where a longest repeated suffix of } X[1]\ldots X[i] \text{ is recognized.}$$

As illustrated in Figure 8, the suffix links index repeated patterns in the sequence and guarantee the existence of common suffixes between the elements that they link. These common suffixes are seen here as musical pasts shared by these elements: $s(i) \in Past_M(i)$. The main postulate of the musical models using the FO is that following non-linear paths using these links creates musical phrases proposing new evolutions while preserving the continuity of the musical discourse, as studied in Assayag and Dubnov [2004].

To summarize, this section extends the heuristics for improvisation, harmonization, and arrangement proposed by Nika and Chemillier [2012] based on the FO navigation proposed in Assayag and Bloch [2007]. The navigation chains paths matching the scenario in the automaton and alternates between linear progressions and jumps following suffix links. The length of the common *context* between two musical events in a sequence is computed in Assayag and Dubnov [2004] by embedding the method introduced in Lefebvre and Lecroq [2000] (linear in time and space) in the construction algorithm. The length of this context corresponds to the parameter c_p (Section 4.1.3) that quantifies the expected "musical quality" of a jump. According to empirical observations made during performances and work sessions with musicians, the subset of $Past_M(i)$ reached by chaining calls to suffix links and reverse suffix links meets the requirements for chaining sequences and digressing from the original sequences.

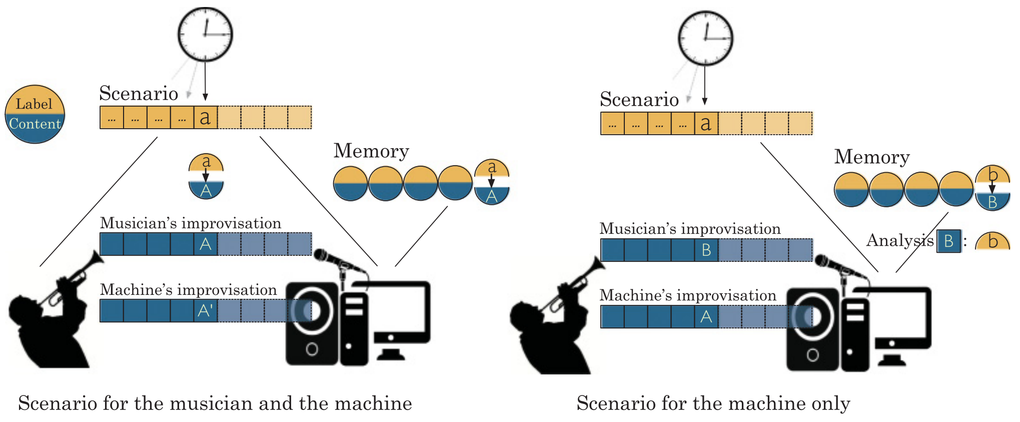

Fig. 9. Possible interactions with the scenario during a performance.

5. AN INTERACTIVE MUSIC SYSTEM: FROM OFFLINE MUSIC GENERATION WITH SPECIFICATION TO ONLINE GUIDED IMPROVISATION

5.1. An Interactive System: Anticipation and Reactivity

This section presents the general architecture of ImproteK used during performances and the interactions with the scenario in a real-time context, as represented in Figure 9. The system plays improvisations matching the scenario, and the music played by the human co-improvisers is captured and added to its musical memory, which can also contain offline material. The incoming stream is segmented using a chosen *external time source* and is annotated by labels. If the scenario is a common referent for the musicians and the machine, then the labels directly come from the scenario (Figure 9, left). If the scenario is seen as a score for the machine only, then the labels come from a chosen analysis (Figure 9, right).

The system reacts to the online control of some *reactive inputs*: the scenario itself and *secondary generation parameters*. These secondary generation parameters correspond to

—the parametrization of the generation model: contraints on the memory region, authorized transformations, maximal/minimal length of the sequences retrieved in the memory, measure of the linearity/non-linearity of the paths in the memory, that is, constraints on the parameters c_f and c_p introduced in Section 4.1.3.
—user-defined content-based constraints to filter the set of sequences in the memory matching the current scenario: for example, pitch interval, onset density, user-defined thresholds or rules, and so on (see examples in **video 3**, Section 5.2.1).

The generic reactive mechanisms are described in Section 5.2 without focusing on where the controls come from: depending on the musical project, they can be given to an operator-musician controlling the system, launched by composed reactivity rules, defined in a higher-level improvisation plan (see Nika et al. [2014]), or plugged into an external listening module.

As introduced previously, ImproteK uses an *external time source* that is used as a clock for the improvisation. This input is generic and can be plugged into a fixed metronome, an irregular time track, or a non-metronomic beat coming from a beat tracking system listening to the musician (the system includes a beat tracking module described in Bonnasse-Gahot [2010]). Section 5.3 presents the audio rendering module of the system that synchronises the live audio re-injections with a non-metronomic beat.

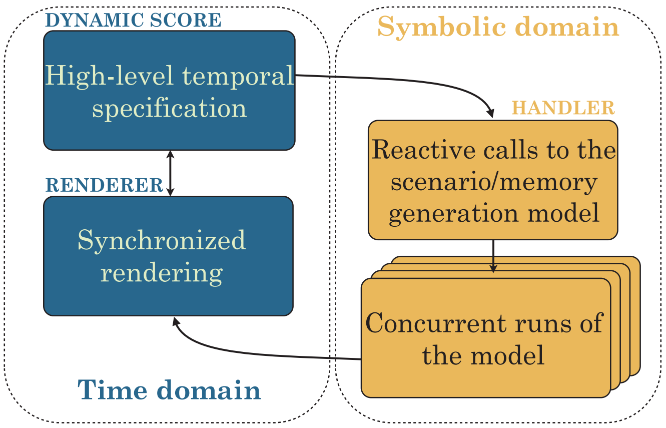
DYNAMIC SCORE
High-level temporal specification
RENDERER
Synchronized rendering
Time domain
Symbolic domain
HANDLER
Reactive calls to the scenario/memory generation model
Concurrent runs of the model

Fig. 10. General architecture of the improvisation system.

5.2. Embedding an Offline Generation Model in a Reactive Architecture to Combine Anticipation and Reactivity

The scenario/memory generation model presented in the previous section is static in the sense that one run produces a whole timed and structured musical gesture satisfying the designed scenario that will then be unfolded through time during performance. It follows a compositional workflow (see Section 6.1) and can therefore be used to generate sequences satisfying given specifications in an offline compositional process. This section details its integration in a dynamic architecture to combine anticipation and reactivity. This architecture manages the dynamic queries sent to the model to generate and refine buffered anticipations in reaction to changes of the reactive inputs introduced in Section 5.1, while maintaining coherence with its long-term horizon.

Figure 10 schematizes the general architecture of the system. Basically, the time domain is where listening, planning, and rendering occur, while the symbolic domain concerns the concurrent runs of the music generation model and the way they are dynamically handled. The architecture articulates three main modules as follows:

—an *improvisation handler* (Sections 5.2.1 and 5.2.2), a *reactive agent* embedding the memory and the mechanisms handling music generation, which manages reaction and concurrency of the overlapping queries sent to the scenario/memory generation model;
—a *dynamic score* (Section 5.2.3), a *reactive program* that manages the high-level temporal specifications and synchronizes the generation and rendering processes with the inputs from the environment;
—an *improvisation renderer* (Section 5.3) synchronizing the rendering of the generated sequences with the environment and informing the dynamic score with time markers.

The generation model and the improvisation handler are implemented in the Open-Music environment [Bresson et al. 2011]. The components involved in time domain are implemented in the graphical programming environment Max [Puckette 1991] using the synchronization strategies of the score follower Antescofo [Cont 2008] and its associated programming language [Echeveste et al. 2013a, 2013b] to write the dynamic score.

5.2.1. Guided Improvisation as Dynamic Calls to an Offline Model: The Improvisation Handler. In the scope of music improvisation guided by a scenario, a *reaction* of the system to dynamic controls cannot be seen as a spontaneous instant response. The main interest of using a scenario is indeed to take advantage of this temporal structure to anticipate the music generation, that is, to use the prior knowledge of what is expected for the future in order to better generate at the current time.

Whether a reaction is triggered by a user control, by hard-coded rules specific to a musical project, or by an analysis of the live output of a musician, it can therefore be considered a revision of the mid-term anticipations of the system in the light of new events or controls. To deal with this temporality in the framework of a real-time interactive software, we consider *guided improvisation* as *embedding an offline process into a reactive architecture*.

In this view, reacting amounts to composing a new structure in a specific timeframe ahead of the time of the performance, possibly rewriting previously generated material. When it is used offline, a whole musical sequence is generated through a single run of the scenario/memory generation algorithm (Section 4.1.4) which chains linearly the successive generation phases. Here, the generation process is segmented and non-linear and involves concurrent generation phases that produce overlapping improvisation fragments. Thanks to the scenario, music is produced ahead of the performance time, buffered to be played at the right time or rewritten. For purposes of brevity (and far from any anthropomorphism),

—*anticipations* will be used to refer to the current state of the already generated musical material ahead of the performance time,
—*intentions* will be used to refer to the current state of the scenario and secondary generation parameters ahead of the performance time.

In this framework, we call *reaction* an alteration of the intentions leading to a call to the generation model to produce a fragment of improvisation starting at a given position in the scenario. The evolving anticipations of the machine result from successive or concurrent calls to the generation model. Introducing a reaction at a time when a musical sequence has already been produced amounts then to rewrite buffered anticipation. The rewritings are triggered when reactive inputs (Section 5.1) are modified, that is, by modifications of the intentions regarding the scenario itself or other generation parameters. The different musical issues raised by these two cases are discussed in Section 6.2. **Video 3** shows the anticipations being rewritten in two offline simulations with different configurations regarding the scenario, the memory, and the chosen reactive inputs.[10]

5.2.2. Concurrent and Overlapping Runs of the Generation Model. The *improvisation handler* is a reactive agent embedding the scenario/memory generation model. It translates the dynamic controls on the reactive inputs into reactive queries for the generation module. A *query* launched by a reaction of the improvisation handler generates an improvisation fragment starting at time q in the scenario.[11] It triggers a run of the generation model to output a sub-sequence (or a concatenation of sub-sequences) of the memory that

—matches the current state of the scenario from date q (i.e., a suffix S_q of the scenario),
—satisfies the current state of the set of generation parameters.

This mechanism can be time-triggered or event-triggered, that is, resulting respectively from depletion of previously generated material or from modifications of the reactive inputs.

[10]**Video 3:** www.youtube.com/watch?v=w8EWxFijB4Y. See the description of the associated artistic collaboration in the online appendix section at the end of the journal.
[11]q is the time at which this fragment will be played; it is independent of the current performance time and the date at which the query is launched by the improvisation handler.

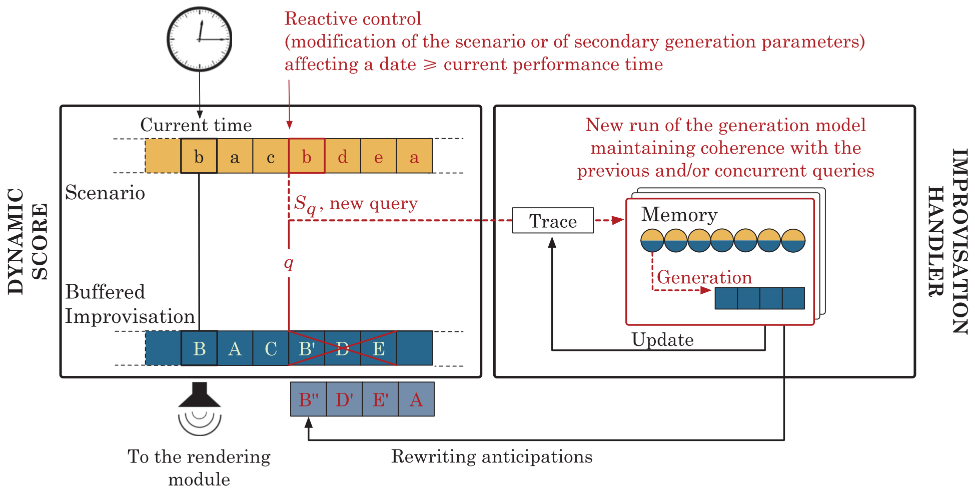

Fig. 11. Reaction seen as rewriting previously generated and buffered anticipations. The live reactions (modifications of the secondary generation parameters or of the scenario itself) trigger concurrent and overlapping queries sent to the offline generation model resulting in the generation of overlapping improvisation fragments. The coherence between these fragments is maintained thanks to an execution trace.

As illustrated in Figure 11, the new improvisation fragment resulting from the generation is sent to the buffered improvisation while the improvisation is being played. The new fragments overwrite the previously generated material on the overlapping time interval. The improvisation handler embeds an execution trace that records the history of the paths in the memory and states of the generation parameters for the last runs of the generation model so coherence between successive generations phases associated to overlapping queries is maintained. This way, the process can go back to an anterior state to ensure continuity at the tiling time q, the first position where the generation phases overlap.

Anticipation may be generated without ever being played because it may be rewritten before being reached by the time of the performance. Similarly, an intention may be defined but never materialized into anticipation if it is changed or enriched by a new event before being reached by a run of generation. Indeed, if reactions are frequent or defined with delays, it would be irrelevant to translate them into as many independent queries leading to numerous overlapping generation phases. We then define an intermediate level to introduce evolving queries, using the same principle for dynamically rewriting intentions as that defined for anticipations. This aspect is dealt with by handling concurrency and working at the query level when the improvisation handler receives new queries while previous ones are still being processed by the generation module. This way, if closely spaced in time queries lead to concurrent processing, relaying their runs of the generation model at the right time using the execution trace enables to merge them into a dynamic query accumulating their features through time. Concurrent queries are thus handled by means of a multi-thread architecture handling concurrent accesses on the shared memory (see Nika et al. [2015]).

5.2.3. Dynamic Score. The dynamic score is a reactive program synchronizing the musical processes with the musical inputs and the control inputs, in particular the external non-metronomic time source introduced in Section 5.1, to adapt to the fluctuating tempo of the human co-improvisers. It is described in detail in Nika et al. [2014]. This reactive program is at the interface between the environment and the improvisation

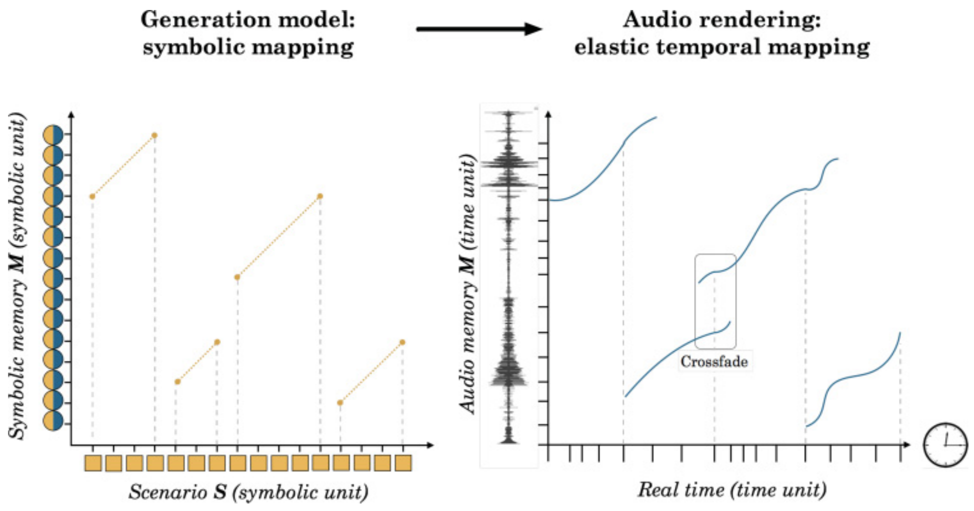

Fig. 12. Generation model: symbolic mapping. Improvisation renderer: time mapping.

handler agent embedding the scenario/memory generation module. The role of the dynamic score is both upstream and downstream since it coordinates the calls to the improvisation handler agent previously described and audio rendering.

On the one hand, it implements parallel processes listening to the musical inputs to segment and label them for learning, reacting to controls, and launching the queries to the improvisation handler. On the other hand, it receives the anticipated improvisations from the scenario/memory generation as portions of code that are executed in due time. The dynamic score acts as a dynamic sequencer: The anticipated improvisation fragments are received and buffered to be unfolded in the real time of the performance and sent to the *improvisation renderer*.

5.3. Audio Rendering and Synchronization with a Non-Metronomic Beat

5.3.1. Live Audio Re-injection. The scenario/memory generation model proceeds to a symbolic mapping between the units of the scenario and that of the memory. The audio renderer presented here proceeds then to the elastic temporal mapping between the symbolic improvisation fragments and the real time of performance.

The musical memory is recorded in an audio buffer, and an index of the time markers corresponding to the events segmented by the external time source is built online in the dynamic score (Section 5.2.3). This way, each unit of the symbolic sequences returned by the generation model is associated to the corresponding dates to read in the buffer. The system can therefore improvise by re-injecting live audio material, which is processed and transformed online to match the scenario, in synchrony with a fluctuating pulse (Figure 12).

Different voices constituting the machine improvisation are defined as different instances of a same generic process running in parallel. This process continuously sends the positions to read in the buffer via a phase vocoder, SuperVP [Depalle and Poirot 1991], whose re-synthesis of sound files can be modified by transformations such as time stretching, pitch shifting, filtering, and so on. When two successive positions of the scenario are mapped to discontiguous slices in the buffer, a crossfade effect is used between a new voice instance and the previous one that is killed only after this relay.

5.3.2. Synchronization with a Non-Metronomic Beat. The synchronization with the environment and a fluctuating pulse is achieved by combining the synchronization strategies

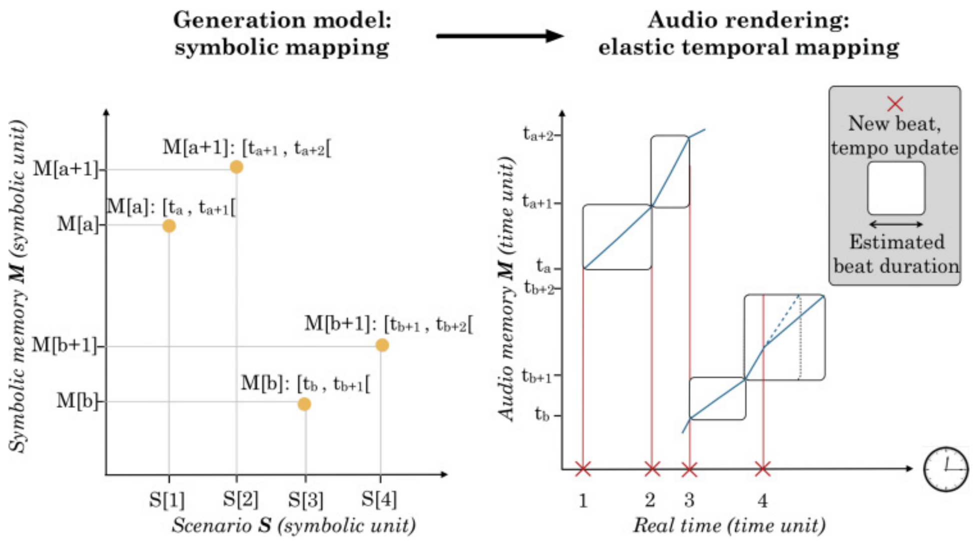

Fig. 13. Tempo estimation, synchronization of the audio rendering with a non-metronomic beat.

of the dynamic score and the phase vocoder that enables time stretching with preservation of the transient signal components [Röbel 2003].

The Antescofo language includes specific variables whose updates are listened to like the musical events coming from a musician in a performance of mixed music using score following. This way, the system can synchronize with the updates of these variables the same way it follows the tempo of a performer. When such a variable is declared with a prior knowledge on the periodicity of its updates, a new tempo estimation is performed every time it is modified using the algorithms introduced in Large [2001] and Large and Jones [1999]. The dynamic adaptation of the speed for reading in the buffer to the real time of the performance is thus done by binding a synchronization variable to the external non-metronomic beat (Figure 13).

Video 4 illustrates the processes managing synchronized live audio re-injections with examples of scat co-improvisations between a musician and the system.[12] For all of these improvisation sessions, the software starts with an empty musical memory and improvises by re-injecting the live audio material, which is processed and transformed online to match the scenario while being reactive to external controls.

6. SCENARII, SCENARIOS, AND "META-COMPOSITION"

In the video examples of the previous sections, the musical purpose of the scenario was to ensure the conformity to the idiom it carried and to introduce anticipation in the generation process. Other musical directions than improvisation in an idiomatic context can be explored using the formal genericity of the couple scenario/memory and the possibility to define dynamic scenarios. Defining scenarios described with other idiomatic vocabularies, audio-musical descriptors, or any user-defined alphabet can lead to approach new dimensions of guided interactive improvisation.

Rowe [1999] outlines that the delegation of some of the creative responsibility to a computer and a performer when designing interactive musical systems pushes up musical composition "to a meta-level captured in the processes executed by the computer"

[12]**Video 4:** www.youtube.com/watch?v=MsCFoqnvAew. See the description of the associated artistic collaboration in the online appendix section at the end of the journal.

and that "an interesting effect of this delegation is that it requires a very detailed specification of the musical decisions needed to produce a computer program at the same time that the composer cedes a large measure of control over musical decision-making to the human improviser." This section underlines the genericity of the scenario/memory approach and how it can be used to add a meta-level of authoring/composition in addition to that constituted by the design of the scenario itself. Section 6.1 sketches a protocol to compose improvisation sessions. In this framework, musicians for whom the definition of a musical alphabet and the design of scenarios for improvisation is part of the creative process can be involved in this "meta-level" of composition, that is, involved upstream to design a part of this "delegation."

6.1. Typology of Alphabets: Idiomatic, Content-Based, User-Defined Arbitrary Alphabets

The generation module used in the system is implemented as a modular library extending this formal genericity in its implementation. It is designed to provide a protocol to compose improvised performances so one can

(1) Define a segmentation unit and a musical alphabet for the labels.
(2) Define the properties of this alphabet, that is, equivalences between the labels. These equivalences can be different for the comparisons memory/memory (involved in learning) and the comparisons scenario/memory (involved in generation).
(3) Define transformation rules between the musical contents belonging to the different equivalence classes.
(4) Compose at the structure level (that is, define a fixed or dynamic scenario).

Figure 14 gives two applications of this protocol with an idiomatic alphabet and a content-based alphabet. The content-based alphabet is illustrated with the example of a vector of chosen audio descriptors. **Video 5** shows an example of improvisation using a composed scenario (without pulse) using such a content-based alphabet.[13] **Video 6** shows an example of offline generation using the analysis of a target audio file as a scenario.[14]

The idiomatic case is illustrated with a harmonic alphabet as in the example given in Figure 3. It should be noted that this category can also cover specific metric structures, clave patterns, musical accents, or any underlying structure materialized or not in the music itself. It generalizes to purely arbitrary user-defined alphabets so a musician can define her/his own grammar associated to specific online or offline musical material. **Video 7** illustrates this point with an extract of improvisation session based on a scenario composed using an abstract alphabet.[15]

Transposition of the musical content or applying gain to the signal as in the examples in Figure 14 are intuitive transformations when the chosen alphabet is respectively harmonic or including a loudness descriptor. In the case of an arbitrary alphabet, this mechanism can be used in a creative way to define equivalences modulo user-defined transformations.

Drawing a distinction between these different alphabets is not only a technical question: Running a generation model using regularities and common patterns in temporal structures leads to different musical results depending on whether these temporal structures describe an underlying formal evolution or the evolution of low-level signal

[13]**Video 5:** www.youtube.com/watch?v=mQeG2e7hPVQ. See the descriptions in the online appendix section at the end of the journal.
[14]**Video 6:** www.youtube.com/watch?v=am_YDsu4Ko8. See the descriptions in the online appendix section at the end of the journal.
[15]**Video 7:** www.youtube.com/watch?v=yW8PkIl4tDE. See the description of the associated artistic collaboration in the online appendix section at the end of the journal.

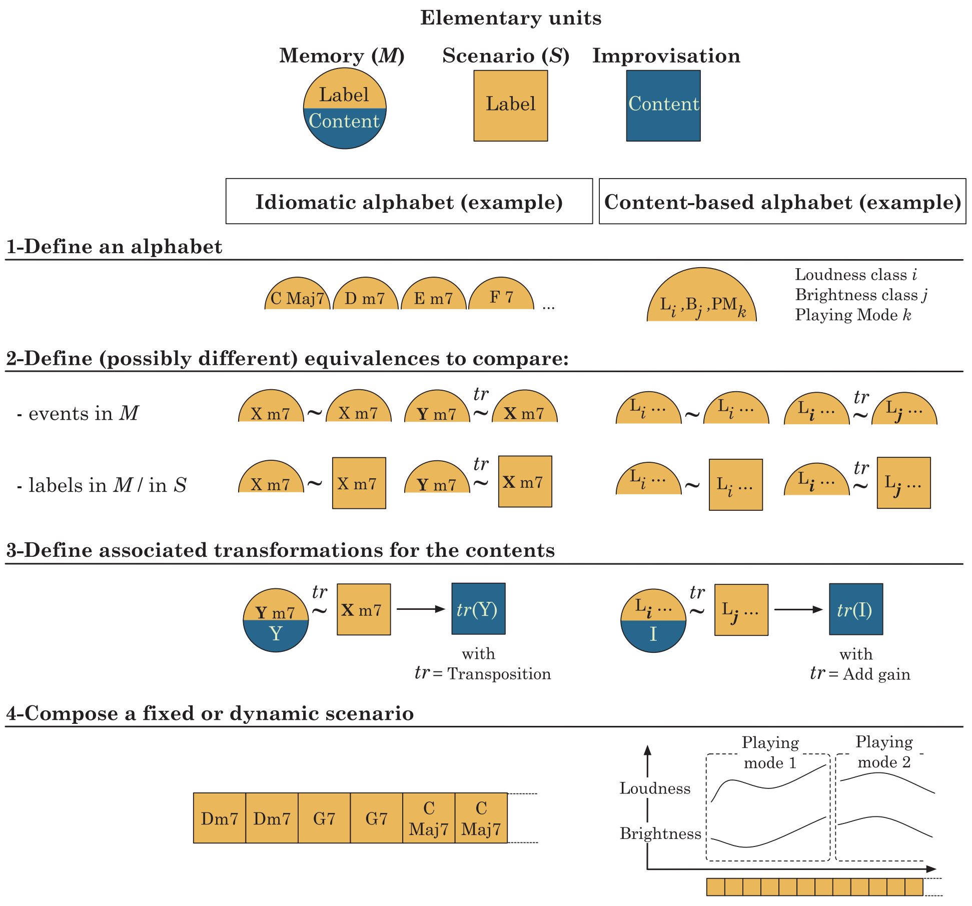

Fig. 14. Towards a protocol to compose improvised performances.

features. Besides, these cases lead to differentiate the musical roles played by the scenario. When the scenario is defined over an idiomatic or arbitrary alphabet describing prior knowledge of an underlying structure, it represents a common referent for all the musicians and the machine. Therefore no analysis mechanisms are needed to label the live musical inputs since the machine shares a common plan with the musicians. On the contrary, in the case of a content-based alphabet, an online or offline analysis is required for learning the memory. The scenario may only describe the part of the machine which improvises without prior knowledge of its musical inputs (**video 5**). The typology of alphabets is thus strongly linked to the typology of scenarios that is completed in the following paragraph.

6.2. Typology of Scenarios: Fixed or Dynamic Scenarios

The articulation between the formal abstraction of scenario and reactivity enables to explore different musical directions with the same objects and mechanisms, providing dynamic musical control over the improvisation being generated. In first approach, we differentiate two playing modes depending on the hierarchy between the musical dimension of the scenario and that of control. When scenario and control are performed on different features of the musical contents, the model combines long-term structure with local expressivity (Section 6.2.1). When scenario and dynamic control act on the same musical feature, it deals with dynamic guidance and intentionality (Section 6.2.2).

6.2.1. Long-Term Structure and Local Expressivity. We first consider the case where the specification of a scenario and the reaction concern different features, conferring them different musical roles (for example, defining the scenario as a harmonic progression and giving real-time controls on density, designing the scenario as an evolution in register and giving real-time controls on energy). In this case, a fixed scenario provides a global temporal structure on a given conduct dimension, and the reactive dimension enables us to be sensitive to another musical parameter. The controlled dimension has a local impact and deals with expressivity by acting at a secondary hierarchical level to filter dynamically the outputs resulting from the research on the fixed dimension (for example, with instant constraints on timbre, density, register, syncopation, etc.). This playing mode may be more relevant for idiomatic or composed improvisation with any arbitrary vocabulary in the sense that a predefined and fixed scenario carries the notions of high-level temporal structure and formal conformity to a given specification anterior to the performance, as it is the case for example with a symbolic harmonic progression.

6.2.2. Guidance and Intentionality. When specification and reaction act on the same musical dimension, the scenario becomes dynamic. A reaction does not consist in dynamic filtering of a set of solutions as in the previous playing mode but in the modification of the scenario itself. In this case, the current state of a dynamic scenario at each time of the performance represents the short-term "intentionality" attributed to the system, which becomes a reactive tool to guide the machine improvisation by defining instant queries with varying time windows. The term "scenario" may be inappropriate in this second approach since it does not represent a fixed general plan for the whole improvisation session. Yet, whether the sequence guiding the generation is dynamic or static (that is, whether the reaction impacts the guiding dimension or another one), both cases are formally managed using the same mechanisms.

7. CONCLUSION

We presented a guided generation model and a reactive architecture introducing authoring and control in an interactive music improvisation process. Both are implemented in the improvisation system ImproteK, whose development process integrates frequent interactions with numerous expert musicians through concerts, work sessions, and filmed listening sessions and interviews (see Nika [2016]).

This system is based on a music generation model guided by a formal scenario. It uses the prior knowledge of this temporal structure to introduce anticipation in the music generation process. In this framework, improvising means articulating this scenario with an annotated online or offline memory, both sequences being represented as words defined on the same alphabet. The generation process follows the scenario while maintaining coherence with the musical logic of the memory by exploiting similar patterns in the sequences as well as their regularities to create new contents beyond simple copy. At the algorithmic level, the current works concern the optimization of the research and their integration into a unified process to deal with the general issue raised by the musical application: Find optimal/creative paths in a text (memory) being guided by a word-constraint (the scenario) and take advantage of regularities to go beyond the concatenations of independent similar patterns. For example, some regularities in the memory provided by its automaton structure are not used in the prefix indexing step of the generation algorithm. Both pattern indexing and research for regularities will therefore be refined to benefit from each other (respectively exploring heuristics based on Crochemore et al. [2013a] and Alstrup et al. [2004]; Crochemore et al. [2013b]).

We presented the embedding of this offline generation module in a reactive framework steered/controlled by external events. This architecture combines dynamic controls and anticipations relative to a predefined or dynamic plan. Concurrent queries launched to the generation module produce short-term anticipations matching the scenario that are refined as the performance goes. The reactive architecture described in this article proposes a model to answer the question “how to react?” but does not address the question “when to react and with what musical response?” Indeed, the model defines the different types of reactions that have to be handled and how it can be achieved. It chooses to offer genericity so reactions can be launched by an operator using customized parameters or by a composed reactivity defining hard-coded ruled specific to a particular musical project. Yet integrating approaches such as reactive listening could enable the system to have reactions launched from the analysis of live musical inputs. As a first step, an architecture was sketched to address the playing mode introduced in Section 6.2.1 [Chemla-Romeu-Santos 2015]: guiding the machine improvisation along two dimensions by associating an anticipation module based on the work presented in this article and a reactive listening module [Bonnasse-Gahot 2014].

Finally, based on the genericity of both the formal model and the implementation, we sketched a protocol to compose improvisation sessions or offline pieces at the scenario level. Thanks to this genericity, future works will include chaining agents working on different musical features. Other perspectives suggest to make use of such procedural and reactive music generation to produce evolving and adaptive soundscapes, embedding it in any environment generating changing parameters while including a notion of plot (for example, in interactive installations or video games, defining a composed dimension using the scenario of the generation model, and an interactive dimension plugged to the reactive inputs of the architecture). As regards musical perspectives, new collaborations will be initiated to experiment with the system in various musical contexts, from improvisation on jazz standards to composed or controlled improvisation situations where defining an alphabet and a grammar is part of the creative process.

ACKNOWLEDGMENTS

We thank Jean Bresson, José Echeveste, Dimitri Bouche, Jean-Louis Giavitto, and Arshia Cont for fruitful discussions and all the musicians who contributed to this project. We also thank the anonymous reviewers for their comments that greatly contributed to improving the quality of this article.

REFERENCES

Andrea Agostini and Daniele Ghisi. 2013. Real-time computer-aided composition with bach. *Contemp. Music Rev.* 32, 1 (2013), 41–48.

Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. 1999. Factor oracle: A new structure for pattern matching. In *SOFSEM 99: Theory and Practice of Informatics*. Springer, 758–758.

Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. 2004. Nearest common ancestors: A survey and a new algorithm for a distributed environment. *Theory Comput. Syst.* 37, 3 (2004), 441–456.

Gérard Assayag. 1998. Computer assisted composition today. In *Proceedings of the1st Symposium on Music and Computers*. Corfu.

Gérard Assayag and Georges Bloch. 2007. Navigating the oracle: A heuristic approach. In *International Computer Music Conference*. Copenhagen, 405–412.

Gérard Assayag, Georges Bloch, and Marc Chemillier. 2006a. Omax-ofon. In *Proceedings of the Sound and Music Computing Conference (SMC'06).*

Gérard Assayag, Georges Bloch, Marc Chemillier, Arshia Cont, and Shlomo Dubnov. 2006b. Omax brothers: A dynamic topology of agents for improvization learning. In *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia*. ACM, 125–132.

Gérard Assayag and Shlomo Dubnov. 2004. Using factor oracles for machine improvisation. *Soft Comput.* 8, 9 (2004), 604–610.

Gérard Assayag, Shlomo Dubnov, and Olivier Delerue. 1999. Guessing the composer's mind: Applying universal prediction to musical style. In *Proceedings of the International Computer Music Conference*. 496–499.

Derek Bailey. 1993. *Improvisation: Its Nature and Practice in Music*. Da Capo Press.

Laurent Bonnasse-Gahot. 2010. Donner à omax le sens du rythme: Vers une improvisation plus riche avec la machine. Ecole des Hautes Etudes en Sciences Sociales, Technical Report.

Laurent Bonnasse-Gahot. 2014. An update on the somax project. Ircam-STMS, Internal Report ANR Project Sample Orchestrator 2, ANR-10-CORD-0018 (2014).

Dimitri Bouche and Jean Bresson. 2015. Planning and scheduling actions in a computer-aided music composition system. In *Proceedings of the Scheduling and Planning Applications Workshop (SPARK'15)*.

Robert S. Boyer and J. Strother Moore. 1977. A fast string searching algorithm. *Commun. ACM* 20, 10 (1977), 762–772.

Jean Bresson, Carlos Agon, and Gérard Assayag. 2011. Openmusic. Visual programming environment for music composition, analysis and research. In *Proceedings of the ACM MultiMedia 2011 (Opensource Software Competition)*.

Jean Bresson and Jean-Louis Giavitto. 2014. A reactive extension of the openmusic visual programming language. *J. Vis. Lang. Comput.* 4, 25 (2014), 363–375.

Marc Chemillier. 2001. Improviser des séquences d'accords de jazz avec des grammaires formelles. In *Proc. of Journées D'informatique Musicale*. Bourges, 121–126. (English summary).

Marc Chemillier. 2004. Toward a formal study of jazz chord sequences generated by steedman's grammar. *Soft Comput.* 8, 9 (2004), 617–622.

Marc Chemillier. 2009. L'improvisation musicale et l'ordinateur. *Terrain* 53, "Voir la musique" (2009), 67–83.

Axel Chemla-Romeu-Santos. 2015. *Guidages De L'improvisation*. Master's thesis. Master ATIAM–Ircam, UPMC.

Arshia Cont. 2008. Antescofo: Anticipatory synchronization and control of interactive parameters in computer music. In *Proceedings of the International Computer Music Conference*.

Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. 2007. *Algorithms on Strings*. Cambridge University Press.

Maxime Crochemore, Lucian Ilie, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Waleń. 2013a. Computing the longest previous factor. *Eur. J. Combin.* 34, 1 (2013), 15–26.

Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. 2013b. Order-preserving suffix trees and their algorithmic applications. *Arxiv:1303.6872* (2013).

Roger Dannenberg. 1989. Real-time scheduling and computer accompaniment. In *Current Directions in Computer Music Research*. Vol. 225–261. MIT Press.

Philippe Depalle and Gilles Poirot. 1991. A modular system for analysis, processing and synthesis of sound signals. In *Proceedings of the International Computer Music Conference*. International Computer Music Association, 161–164.

Alexandre Donzé, Rafael Valle, Ilge Akkaya, Sophie Libkind, Sanjit A. Seshia, and David Wessel. 2014. Machine improvisation with formal specifications. In *Proceedings of the 40th International Computer Music Conference (ICMC'14)*.

Shlomo Dubnov, Gérard Assayag, and Ran El-Yaniv. 1998. Universal classification applied to musical sequences. In *Proceedings of the International Computer Music Conference*. 332–340.

José Echeveste, Arshia Cont, Jean-Louis Giavitto, and Florent Jacquemard. 2013a. Operational semantics of a domain specific language for real time musician–computer interaction. *Discr. Event Dynam. Syst.* (2013), 1–41.

José Echeveste, Jean-Louis Giavitto, and Arshia Cont. 2013b. *A Dynamic Timed-Language for Computer-Human Musical Interaction*. Research report RR-8422. INRIA. Retrieved from http://hal.inria.fr/hal-00917469.

Alexandre R. J. François, Isaac Schankler, and Elaine Chew. 2013. Mimi4x: An interactive audio–visual installation for high–level structural improvisation. *Int. J. Arts Technol.* 6, 2 (2013), 138–151.

Daniel J. Fremont, Alexandre Donzé, Sanjit A. Seshia, and David Wessel. 2014. Control improvisation. *Arxiv:1411.0698* (2014).

Fiammetta Ghedini, François Pachet, and Pierre Roy. 2016. Creating music and texts with flow machines. In *Multidisciplinary Contributions to the Science of Creative Thinking*. Springer, 325–343.

D. E. Knuth, J. H. Morris, and V. R. Pratt. 1977. Fast pattern matching in strings. *SIAM J. Comput.* 6, 2 (1977), 323–350.

Edward W. Large. 2001. Periodicity, pattern formation, and metric structure. *J. New Music Res.* 30, 2 (2001), 173–185.

Edward W. Large and Mari Riess Jones. 1999. The dynamics of attending: How people track time-varying events. *Psychol. Rev.* 106, 1 (1999), 119.

Arnaud Lefebvre and Thierry Lecroq. 2000. Computing repeated factors with a factor oracle. In *Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms*. 145–158.

A. Lefebvre, T. Lecroq, and J. Alexandre. 2002. Drastic improvements over repeats found with a factor oracle. In *Proceedings of the 13th Australasian Workshop on Combinatorial Algorithms*, E. Billington, D. Donovan, and A. Khodkar (Eds.). 253–265.

Benjamin Lévy, Georges Bloch, and Gérard Assayag. 2012. Omaxist dialectics. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 137–140.

Julian Moreira, Pierre Roy, and François Pachet. 2013. Virtualband: Interacting with stylistically consistent agents. In *Proceedings of the of International Society for Music Information Retrieval Conference*. Curitiba.

J. H. Morris and V. R. Pratt. 1970. *A Linear Pattern-matching Algorithm*. Report 40, Computing Center, University of California, Berkeley, 1970.

Jérôme Nika. 2016. *Guiding Human-computer Music Improvisation: Introducing Authoring and Control with Temporal Scenarios*. PhD Thesis. Université Paris 6 Pierre et Marie Curie. Retrieved from https://hal.inria.fr/tel-01361835.

Jérôme Nika, Dimitri Bouche, Jean Bresson, Marc Chemillier, and Gérard Assayag. 2015. Guided improvisation as dynamic calls to an offline model. In *Proceedings of the Sound and Music Computing Conference (SMC'15)*.

Jérôme Nika and Marc Chemillier. 2012. Improtek: Integrating harmonic controls into improvisation in the filiation of omax. In *Proceedings of the International Computer Music Conference (ICMC'12)*. 180–187.

Jérôme Nika and Marc Chemillier. 2014. Improvisation musicale homme-machine guidée par un scénario temporel. *Techn. Sci. Inf.* 33, 7–8 (2014), 627–650.

Jérôme Nika, José Echeveste, Marc Chemillier, and Jean-Louis Giavitto. 2014. Planning human-computer improvisation. In *Proceedings of the International Computer Music Conference*. 1290–1297.

François Pachet and Pierre Roy. 2011. Markov constraints: Steerable generation of Markov sequences. *Constraints* 16, 2 (2011), 148–172.

François Pachet, Pierre Roy, Julian Moreira, and Mark d'Inverno. 2013. Reflexive loopers for solo musical improvisation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.

Jeff Pressing. 1984. Cognitive processes in improvisation. *Adv. Psychol.* 19 (1984), 345–363.

Miller Puckette. 1991. Combining event and signal processing in the MAX graphical programming environment. *Comput. Music J.* 15 (1991), 68–77.

Mathieu Ramona, Giordano Cabral, and François Pachet. 2015. Capturing a musicians groove: Generation of realistic accompaniments from single song recordings. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 4140–4141.

Axel Röbel. 2003. A new approach to transient processing in the phase vocoder. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFx'03)*. 344–349.

Robert Rowe. 1999. The aesthetics of interactive music systems. *Contemp. Music Rev.* 18, 3 (1999), 83–87.

L. Henry Shaffer. 1980. 26 analysing piano performance: A study of concert pianists. *Adv. Psychol.* 1 (1980), 443–455.

George Sioros and Carlos Guedes. 2011a. Complexity driven recombination of MIDI loops. In *Proceedings of the International Society for Music Information Retrieval (ISMIR'11)*. 381–386.

George Sioros and Carlos Guedes. 2011b. A formal approach for high-level automatic rhythm generation. *Proceedings of the Bridges Conference 2011*. University of Coimbra, Coimbra, Portugal.

John A. Sloboda. 1982. Music performance. *The Psychology of Music* (1982), 479–496.

Greg Surges and Shlomo Dubnov. 2013. Feature selection and composition using pyoracle. In *Proceedings of the 9th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Cheng-i Wang and Shlomo Dubnov. 2014. Guided music synthesis with variable markov oracle. In *Proceedings of the 10th Artificial Intelligence and Interactive Digital Entertainment Conference* (2014).

Online Appendix to:
ImproteK: Introducing Scenarios into Human-Computer Music Improvisation

JÉRÔME NIKA, IRCAM STMS Lab (CNRS, UPMC, Sorbonne Universités)
MARC CHEMILLIER, Cams, Ecole des Hautes Etudes en Sciences Sociales
GÉRARD ASSAYAG, IRCAM STMS Lab (CNRS, UPMC, Sorbonne Universités)

The videos cited in this article can be found in a dedicated channel[15] or at vimeo.com/jeromenika. Brief descriptions of the videos are as follows:

Video 0: Compilation of Different Interaction Situations

***Video 0**, cited in Section 1: www.youtube.com/watch?v=OhXf-1QpqEI (or vimeo.com/jeromenika/improtek-compilation).*

Short excerpts of the music improvisation sessions cited in the article.

Video 1: Conformity to an Idiomatic Structure, Improvisation on a Simple Chord Progression

***Video 1**, cited in Section 3.2: www.youtube.com/watch?v=w17pFvrI06A (or vimeo.com/jeromenika/improtek-fox-rentparty).*

Sax improvisations on a simple chord progression using the music improvisation software ImproteK. Work session with Rémi Fox rehearsal for a performance at Montreux jazz festival. The software starts with an empty musical memory and improvises by reinjecting the live audio material that is processed and transformed online to match the scenario while being reactive to external controls. The scenario is the chord progression of “Rent Party” (Booker T. Jones) segmented by beat:

|| : Cm7 Bb7 | AbMaj7 Bb7 : ||[16].

Video 2a: Example of “Hybridization” with an Early MIDI Version of the System

***Video 2**, cited in Section 3.3: www.youtube.com/watch?v=yY3B5qfFri8 (or vimeo.com/jeromenika/improtek-lubat-early).*

Excerpt of a concert with Bernard Lubat. Co-improvisation using an early MIDI version of the system playing theme / variations and a chorus. The musical memory used by the system is constituted by the captured live MIDI material and a very heterogeneous offline corpus (recordings of more than 10 jazz standards or ballads by different interprets). The scenario is the chord progression of “D'ici d'en bas” (Bernard Lubat) segmented by beat:

||: (Fm7 | G7 | Cm7 | Cm7 | F7 | G7 | Cm7 | Cm7)*2
| Fm7 | Bb7 | EbMaj7 | AbMaj7 | D7 | G7 | Cm7 | C7 |
| Fm7 | Bb7 | EbMaj7 | AbMaj7 | D7 | G7 | Cm7 | C m7 :||

[15]Youtube channel “Jérôme Nika”: www.youtube.com/channel/UCAKZIW0mMWCrX80yS96ZxAw.
[16]In this chord progression and in the following: |...| = a bar = 4 beats.

Video 2b: Example of Audio "Hybridization": Trio Holiday, Schwarzkopf, Piaf

Video 2*, cited in Section 3.3*: *www.youtube.com/watch?v=reJ-SiblCcs (or vimeo.com/jeromenika/improtek-sellin-themanilove1-finale).*

"The Man I Love #1" improvisation by Hervé Sellin (piano) and Georges Bloch (using ImproteK). The scenario provided to the system is the chord progression of the song, and its musical memory is

—Hervé Sellin playing "The Man I Love,"
—Billie Holiday singing "The Man I Love,"
—Edith Piaf singing "Mon dieu" and "Milord,"
—Elisabeth Schwarzkopf singing "Mi trad quell'alma ingrata" (Mozart, Don Giovanni) and "Tu che del gel sei cinta" (Puccini, Turandot).

Video 3: In-Time Reaction as Rewriting the Anticipations

Video 3*, cited in Section 5.2.1: www.youtube.com/watch?v=w8EWxFijB4Y (or vimeo.com/jeromenika/improteksmc15).*

Simulations "behind the interface": focus on the improvisation handler agent embedding the offline scenario/memory generation model in a reactive framework. This video shows the anticipations being rewritten when the chosen reactive inputs (see Section 5.1) are modified.

Example 1: scenario: harmonic progression ("Autumn Leaves"), memory: heterogeneous MIDI corpus (captured solos on various blues or jazz standards), chosen reactive dimensions: register and density. *Example 2:* scenario: spectral centroid profile, memory: audio file, chosen reactive dimensions: scenario itself and memory region.

Video 4: "Scat" Co-improvisations, Synchronized Audio Rendering

Video 4*, cited in Section 5.3.2: www.youtube.com/watch?v=MsCFoqnvAew (or vimeo.com/jeromenika/improtek-lubat-scat).*

Compilation of "scat" co-improvisations with Bernard Lubat and Louis Lubat. For all of these improvisation sessions, the system starts with an empty musical memory and improvises by re-injecting the live audio material that is processed and transformed online to match different idiomatic scenarios while being reactive to external controls and synchronized with a non-metronomic beat. The scenarios used in the different examples are metric structures and/or harmonic progressions. In particular, the scenario of the last session presented in the video (1′36) is the chord progression of "J'aime pour la vie" (Bernard Lubat) segmented by beat:

||: (D7 | D7 | D7 | D7)*4 | (G7 Ab7 | G7 F7 | G7 Ab7 | G7 F7)*2 :||

Video 5: Interactive Improvisation with a Composed Scenario—Content-Based Alphabet

Video 5*, cited in 6.1: www.youtube.com/watch?v=mQeG2e7hPVQ (or vimeo.com/jeromenika/improtek-agnes-composed).*

First technical experiments with the composer-improviser Michelle Agnes Magalhaes, who works on structured improvisation. The chosen content-based alphabet is a 3-uple: loudness, brightness, playing mode. This example illustrates the case where the scenario only describes the part of the machine improvisation (see Section 6.1). The system re-injects the live audio material matching the descriptor profiles imposed by the scenario that is composed in such a way that the machine improvisation alternates between counterpoint and extension of the musical gesture.

Video 6: Using the Analysis of a Target Audio File as Scenario

Video 6*, cited in Section 6.1: www.youtube.com/watch?v=am_YDsu4Ko8 (or vimeo.com/jeromenika/improtek-starwospheres).*

A short offline example. The content-based scenario is the profile of spectral centroid and roughness extracted from the soundtrack of a musicless movie scene segmented by audio event. It is applied to a memory constituted by the piece “Atmospheres” (Ligeti) analyzed with the same couple of audio descriptors. The generated sequence replaces the original soundtrack.

Video 7: Generative Improvisation with a Composed Scenario—Abstract Alphabet

Video 7*, cited in 6.1: www.youtube.com/watch?v=yW8PkIl4tDE (or vimeo.com/jeromenika/improtek-fox-generative1).*

Structured improvisation, work session with Rémi Fox, rehearsal for a performance at Montreux jazz festival. The software starts with an empty musical memory and improvises several voices by reinjecting the live audio material that is processed and transformed online to match the composed scenario while being reactive to external controls. The scenario defines two voices (“accompaniment” or “solo”) and an abstract structure segmented by beat:

$$||{:}\ A_1\ B_1\ B_2\ A_1\ B_2\ {:}||$$

with

$A_1 = ||\ \mathrm{X}\ |\ \mathrm{X}^{+5}\ |\ \mathrm{X}^{-2}\ |\ \mathrm{X}^{+3}\ ||$
$A_2 = ||\ \mathrm{X}\ |\ \mathrm{X}\ |\ \mathrm{X}^{+5}\ |\ \mathrm{X}^{+5}\ |\ \mathrm{X}^{-2}\ |\ \mathrm{X}^{-2}\ |\ \mathrm{X}^{+3}\ |\ \mathrm{X}^{+3}\ ||$
$B_1 = ||\ \mathrm{Y}\ \mathrm{Z}\ |\ \mathrm{Z}^{+5}\ \mathrm{X}^{+3}\ |\ \mathrm{Y}\ \mathrm{X}^{+5}\ |\ \mathrm{Z}^{+5}\ \mathrm{X}^{+3}\ |\ \mathrm{Y}\ \mathrm{X}^{-4}\ |\ \mathrm{Y}^{+3}\ |\ \mathrm{Z}^{-5}\ \mathrm{Z}\ |\ \mathrm{Z}^{+5}\ \mathrm{X}^{+3}\ ||$
$B_2 = ||\ \mathrm{Y}\ \mathrm{Z}\ |\ \mathrm{Z}^{+5}\ \mathrm{X}^{+3}\ |\ \mathrm{Y}\ \mathrm{X}^{+5}\ |\ \mathrm{Z}^{+5}\ \mathrm{X}^{+3}\ |\ \mathrm{Y}\ \mathrm{X}^{-4}\ |\ \mathrm{Y}^{+3}\ |\ \mathrm{Z}^{-5}\ \mathrm{Z}\ |\ \mathrm{Z}^{+5}/\mathrm{X}^{+3}\ \mathrm{Y}\ ||$,

where X, Y, and Z are abstract equivalence classes and the exponents represents transposition in semitones. A constraint is added to the “accompaniment” voice to get a repetitive structure: Its memory is restricted to A_1 and the first measures of B_1.