Improtek: integrating harmonic controls into improvisation in the filiation of OMax

Jérôme Nika, Marc Chemillier

IMPROTEK, INTEGRATING HARMONIC CONTROLS INTO IMPROVISATION IN THE FILIATION OF OMAX

Jérôme Nika

Ircam - Paris, then Télécom ParisTech
46 rue Barrault - 75013 Paris
jerome.nika@telecom-paristech.fr

Marc Chemillier

Centre d'analyse et de mathématique sociales
EHESS, 190 avenue de France - 75013 Paris
chemilli@ehess.fr

ABSTRACT

We introduce ImproteK, a system integrating a rhythmic framework and an underlying harmonic structure in a context of musical improvisation. In the filiation of the improvisation software OMax [4, 3, 13], it is built on the factor oracle structure to take advantage of the particularly relevant and rich characteristics of this automaton in a musical environment [5]. Moreover, it can adapt to a regular beat and produce improvisations following a given chord progression. ImproteK is conceived as an interactive instrument dedicated to performance: its improvisations are based on the style modeling performed on live playing or on an offline corpus. Combined with pattern reuse techniques, this modeling expands on harmonization and arrangement in a harmonic interaction module.

1. INTRODUCTION

The aim of ImproteK is to combine style modeling and interaction to install an original dialogue between musicians and a virtual improviser feeding its inspiration on their playing. Both these paradigms and the automaton structure at the heart of its implementation (section 2) make this system a cousin of the improvisation software OMax [4, 3, 13] conceived and developed at Ircam.

Following previous works on musical style modeling by G. Assayag et. al [6], OMax is capable of learning the style of a human improviser thanks to a representation based on the oracle structure introduced by C. Allauzen et. al [1] extended to a musical context [5]. The software builds a model of a musician's playing in real-time, and is then able to navigate through this representation by following different paths from that taken by the musician. This leads to generate original improvisations with a common aesthetics.

With the same intention, ImproteK focuses on measured music supported by an underlying harmonic structure. It provides an enriched interaction by taking the beat into account in the framework of a given chord progression (section 3). This conception of improvisation is made concrete by an architecture enabling its integration in a musical band whose tempo can be extracted and followed (section 4). Finally, it introduces harmonic interaction by extending style modeling to the scope of harmonization and arrangement (section 5).

The purpose of this paper is first to recall some general principles underlying both ImproteK and OMax approaches, and then to describe the specific developments of the ImproteK software.

2. MUSICAL STYLE MODELING AND ORACLE STRUCTURE

2.1. Style modeling in improvisation and harmonization

Since M&Jam Factory [22], R. Rowe's Cypher [19], or G. Lewis' Voyager [14], often considered as the first real-time interactive systems, many "virtual improvisation partners" have been conceived. Most of them benefited from the development of machine learning techniques with the growing idea to get always closer to the interacting human performer's discourse.

Among them, the Continuator [16] conceived by F. Pachet models the musical input using an extended Markovian model to create new phrases from this learning. As in OMax *free* mode, there is no rhythmic perception to enable a synchronization with the musician. On the other hand, GenJam [7] developed by J. Biles is closer to the previous experiments of OMax with a *beat* mode: the software provides an accompaniment with a given tempo to support a musician's improvisation. After listening, it repeats some sequences modified through a genetic algorithm. B. Thom's Band Out of a Box [21] also involves a non-interactive computer accompanist with a fixed tempo in a trading fours interaction scheme where a human improviser and a virtual partner repeatedly call and respond in four-bar chunks. Each bar of the human improvisation is assigned to a cluster called *playing mode*, and the computer response is constituted by 4 bars belonging to the same sequence of modes.

This last example introduces the recurring issue of segmenting and representing the musical inputs. This point is particularly crucial in the corpus-based systems dealing with harmonization, arrangement or accompaniment such as ImproteK. Whether they are interactive and dedicated to performance or conceived to be used offline, these systems can basically be classified according to two characteristics: the exclusive use of formalized musical rules and the use of a musical corpus (see [15] for the positioning of ImproteK within a more exhaustive classification system).

In this last category, the corpus can be considered as a training environment to create generative models (among them C. Chuan and E. Chew [11], Microsoft Mysong-Songsmith software [20]) and/or as a musical memory in which fragments are retrieved and combined to create the new material (G. Ramalho's jazz bass player ImPact [18]).

The definition of the segmentation unit in the corpus processing is one of the key differences: the *grain* can be key notes of a reduced melody in C. Chuan and E. Chew's system, a single chord in Songsmith and the software Band in a Box (PG Music), or particular chord chunks in G. Ramalho's ImPact. The idea is to find the right balance between long enough slices to provide plausibility and coherence in the returned accompaniment, and fine enough slices to avoid recopying too long and identifiable segments from the corpus. In the case of Improtek, the chosen unit is the beat. Indeed, the oracle structure prevents from choosing between musical smoothness and innovation by providing continuity by construction, and enables to work with such a fine grain.

2.2. Oracle and pattern recognition

The oracle was initially conceived for optimal string matching, and was extended for computing repeated factors in a word and for data compression. This acyclic automaton represents at least all the factors in a word, and the incremental construction algorithm is time and space linear in its length (for the details of the construction algorithm, see [1]).

The formal properties of the oracle structure are detailed in the founding articles, and fully applied to the issue of *stylistic reinjection* [4] in several OMax papers. We briefly summarize here the tools providing the continuity and coherence of the sequences generated through an oracle: the forward transitions and the suffix links.

The *forward transitions* (forward links, plain lines) enable to reach every sub-pattern in the original sequence starting from the initial state. In this way, every progression between consecutive states of the original sequence can be generated.

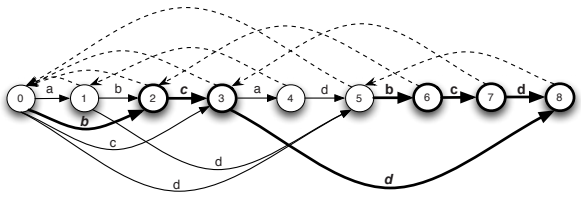

Figure 1. Oracle for the sequence *abcadbcd*: generation of a sub-pattern starting from the initial state.

Fig. 1 shows the oracle for the sequence *abcadbcd* and illustrates this point by displaying a path using forward transitions to generate the sub-sequence *bcd* (0, 2, 3, 8), originally occurring between the states 5 and 8.

The oracle also locates the repeated sub-pattern in the original sequence with the *suffix links* (backward links, dashed lines). They point at the final state of the previously encountered occurrences of a pattern.

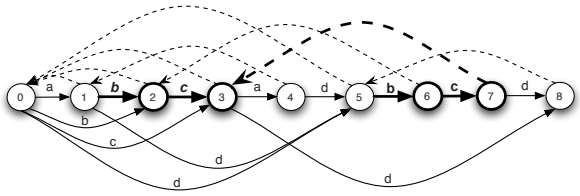

Figure 2. Oracle for the sequence *abcadbcd*: repeated sub-pattern.

In the example of fig. 2, a suffix link connects the states 7 and 3. 3 is indeed the leftmost position where a longest repeated suffix of the sequence ending in 7 (*abcadbc*) is recognized: *bc*.

The oracle structure can be used as a tool to create new sequences by a non-linear navigation following the previously introduced links thanks to its understanding of the logic in the progressions present in the strings. In a musical application, its ability to preserve the discourse of the original material enables to develop an aesthetics close to that of the musicians in interaction.

3. IMPROVISING WITHIN A METRIC AND HARMONIC FRAMEWORK

3.1. The beat hypothesis

The philosophy of the current version of OMax is to turn every single musical event into a state in an oracle object. The *free* real-time navigation [2] through a thus structured memory makes it an interactive instrument dedicated to improvisation in a *free* musical context. The approach of the ImproteK project differs by the integration of two paradigms. First, its improvisations take place in an underlying harmonic structure represented by a chord progression we will refer to as *grid*. In this way, it makes the system able to expand its modeling on harmonization and arrangement. Then, it takes a metric framework into account by setting the beat as the elementary unit in its acquisitions, restitutions, and generations.

The only musical hypothesis is therefore the existence of a regular beat in the material listened and produced by ImproteK. The tendancy to the synchronization with a periodic beat being a deep universal of the human music perception [17], this sole assumption does not make the system oriented towards a restricted musical field. Starting from this point, we add a notion of *labels* making some musical slices separated by beats, making them *equivalent* (see section 5.1.2). This process enriches the set of possible combinations, but does not carry any harmonic hypothesis. Indeed, as discussed in the last section, the references to the jazz idiom through this article come from the context in which ImproteK has been used so far.

3.2. *Mode beat* oracles

The improvisation module is built on a previous version of OMax (OMax 2.0, 2004) conceived as a Lisp library under the OpenMusic environment [9]. This version implemented the oracle structure both in a free improvisation context (mode *free*, direction adopted by the current version of OMax) and with a regular beat (mode *beat*, starting point of ImproteK).

In this view, each state of the oracles represents a musical slice whose duration is given by the current tempo and which contains differents types of events happening between two beats. These events can be musical MIDI sequences - melody or accompaniment fragments - or symbolic information such as chord labels. During the generation steps, these different features constituting the states will sometimes be considered as elementary outputs concatenated in a *fragment reuse* process, and sometimes seen as labels to compare with the given path to follow in the navigation through the oracle.

3.3. Learning and improvising with the *live oracle*

ImproteK improvises by retrieving and combining preexistant elementary units: the new phrases are built by concatenating "beat slices" coming from its musical memory. These fragments are not independently and randomly drawn but continuously collected by following the chosen harmonic grid supporting the musical session as a guideline.

This process involves a first instance of the oracle structure: the *live oracle*, which carries its learning process out on the phrases played by the musicians (fig. 3). These MIDI inputs are indeed indexed beat by beat in real-time by the chord labels of the current harmonic grid (see section 4.2) and are therefore formated for the building of this object.

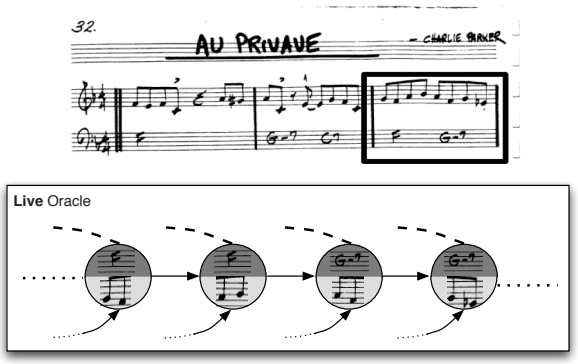

Figure 3. Learning the *live oracle*

What we call "improvise" is the generation of new musical phrases using the memory stored in this live oracle. To do so, the chord labels in the oracle are considered as indexes in the context of a *constrained navigation*, a heuristic designed by G. Assayag. This mechanism amounts to searching sub-patterns of an input sequence (here the harmonic grid) in a different sequence (here the oracle's harmonic labels). When a label matches, the melodic beat slice present in the state is returned to be added to the the sequence constituted by the results of the previous steps.

3.4. Constrained Navigation and continuity

To "improvize following a grid" means here to follow a path in the automaton using transitions indexed by labels coming from that grid. At each step, if no matching label is found from the current state, the navigation tries to follow a suffix link pointing on an other state where the required label could be read. The suffix links provide the existence of a common context between both concatenated fragments in the orignal sequence used to build the oracle. In this way, the navigation first looks for continuity by trying to stick to the previously learned progressions. Then, it searches for the labels independently of the local context if they do not appear, even after transposition, in the chosen oracle.

A *continuity parameter* is tested at each step of the calculation. It counts the number of successive forward transitions followed during the navigation, and then gives the length of the duplicated segments from the sequence in the oracle. By imposing a maximum continuity parameter, one can therefore quantify the wished balance between fidelity to the original sequence and originality in the generated improvisation: a high value will lead to a high ressemblance whereas a low value will bring more surprises.

The constrained navigation process consists in reading the successive labels of the input grid to look for beats indexed by these same labels in the oracle. If the current continuity parameter does not exceed the imposed maximum continuity, the search is operated following graded modes:

- *Continuity mode*: If its label matches, follow a forward transition, update the current beat position in the oracle, and output the associated melody fragment. If not, switch to *Suffix mode* if operating suffix links are found, otherwise switch to *Nothing mode*.
- *Suffix mode*: Follow the suffix link pointing on the longest repeated suffix to reach a matching label (see fig. 4) , update the current beat position in the oracle, and output the associated melody fragment. Otherwise switch to *Nothing mode*.
- *Nothing mode*: The pattern matching is performed independently of the context and the label is searched in the whole oracle. A transposition can be used if necessary.

Fig. 4 shows an example of generation step. At the current stage, the chord labels *d*, *b* and *c* have been searched and found in the *live oracle*, and the concatenation of the associated musical fragments *D*, *B´* and *C´* forms the

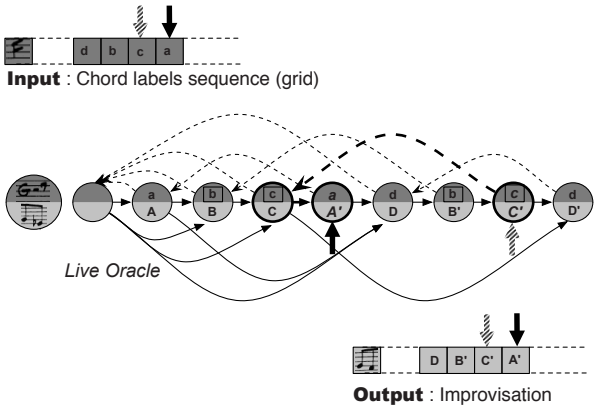

Figure 4. Constrained navigation through the live oracle

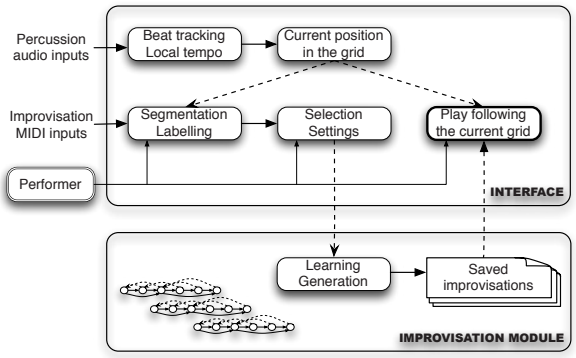

Figure 5. Architecure

phrase being built. The current position in the oracle is the next-to-last state (*c / C´*) and the following chord label read in the input grid is *a*. No forward transition pointing on a matching label can be found, so we switch to *suffix mode*. The suffix link starting from the current state points, by definition, on the final state of the lefmost repeated suffix of *dbc* (*bc*): (*c / C*). A forward transition matching the searched label *a* is found in this state. So we finally follow the suffix link to jump from (*c / C´*) to (*c / C*) where we arrive in the common context (*bc*) to be able to reach the new beat (*a / A´*)

The notion of common context is crucial because it ensures coherence and smooth musical transitions in the generated sequences even if the navigation involves jumps between different zones of the oracle. Indeed, in the case of this exemple, neither the input chord labels sequence nor the output musical segment can be found on their own in the memory of the live oracle.

4. TOWARDS AN INTERACTIVE INSTRUMENT

4.1. Architecture

The live performance from the musicians is received, segmented, formatted and saved in real-time via an interface developed under Max/MSP. Through this same interface, the ImproteK user selects a symbolic harmonic grid which constitutes the spinal column of the improvisation. Then, he has access to control parameters to guide the calculation of different kinds of musical phrases: new improvisations accompanied or not (see section 5) or live arrangements of the grid (see paragraph 4.5).

These processing instructions are sent via OSC protocol to the OpenMusic improvisation module introduced in the previous section (fig. 5), following therefore the global architecture of OMax 2.0 who worked in live interaction with Max/MSP through a basic interface who did not deal with the tempo management exposed in the following paragraph.

Under this version, a metronymic beat was imposed by the computer. This condition necessarily implied the loss of a part of the musical richness, and constrained the musician's expression. Based on this analysis, a tempo following module was implemented to extract the tempo directly from the current improvisation session.

4.2. Get rhythm

During the performance, an external source of beat is required to segment the inputs from the musician in the listening process, and to play phrases rhythmically fitting the current improvisation. To do so, ImproteK integrates the association of a *beat tracker* and a *score follower* acting as a sequencer by emitting in real-time the current position in the harmonic grid. First, it marks every beat of the listened live improvisations with the current chord label. Then, the phrases returned by the improvisation module being labelized sequences, it triggers the playing of the corresponding beat slices.

The aim of the ImproteK project is to be a proper interactive instrument finding its place within a band. It was therefore fundamental that its use did not limit creativity and expressivity by subordinating the musicians to a metronomic tempo dictated by its outputs. The Max/MSP *beat tracker* object [8] was especially developed to be integrated into an environment related to OMax to result in a richer musicality, the previous applications of the *mode beat* oracles involving a fixed tempo imposed by the system.

This module processes MIDI or audio streams (for example the rhythm section of the band) and continually estimates the local tempo to be used to generate and play improvisations adjusted to the necessarily variable human beat. During an initialization phase, the user provides a tempo indication with a manual time beating whose goal is also to give the initial phase by indicating the absolute dates of the beats in order to avoid an offbeat synchronization. Following this stage, the tempo variations are calculated throughout the whole performance (the description of the beat tracker is not the subject of the present paper, more details can be found in [8]). This signal is then used as a clock to trigger the different elements constituting the sequences loaded in Antescofo.

4.3. Using a score follower as a sequencer

Antescofo [12] is a polyphonic score following system and a synchronous programming language for musical composition conceived by A. Cont. This object, developed as an external module for Max/MSP and PureData programming environments, conducts an automatic recognition of music score position from a real-time audio stream. It enables the synchronization of an instrumental performance with the computer realized elements of an electronic score.

The sequences calculated by the Lisp/OpenMusic module are written and saved as Anstescofo scores. The phrases thus generated broaden the improvisations collection and are available to be loaded by the performer during the improvisation: in our use of Antescofo, the beats provided by the beat tracker object act as the ”notes” in the electronic score, and the contents to be triggered are the beat slices. Finally, an improvisation generated from an acquisition performed at a given tempo can be played with a different one since the time notation under the Antescofo format is relative.

4.4. Segmenting and indexing the inputs

This couple of objects in charge of the beat management is found downstream to play the generated sequences, and equally upstream in the live acquisition process. The improvisation session takes place in the scope of a known harmonic grid, and this last is written as a progression of chord labels, each of them being associated to a numbered beat under the Antescofo format. In this way, a live indexing of the MIDI inputs can be performed with these harmonic labels.

We use a particular encoding to mark the MIDI streams: the channel 16 is dedicated to the grid representation with conventions to notify the root and the type of the chords which are received from Antescofo. In this way, the MIDI buffers used by the improvisation module contains all the required information of segmentation and indexing to build new oracles or musical phrases.

4.5. A source of proposals

ImproteK is conceived as an instrument played by a full-time performer interacting with the rest of the band. He pilots the system through a graphical interface and controls in real-time the parameters of both steps of learning and generation which are initiated on demand. During the performance, he selects the musical phrases given as inputs for the style modeling, determines the current oracles (the live oracle as well as the instances involved in the harmonization and arrangement module introduced in section 5) and sets technical parameters such as the maximum continuity parameter in the navigation for every oracle. Among other characteristics, the learning material, the length, or even the ”boldness” of the improvisations are therefore left to his discretion.

The performer is not only in charge of the listening and learning but he also fully leads the playing via keyboard controls. He has access to phrases generated from the last musical events as well as the ones from the previous sessions. Different forms of improvisation are available: they can be for instance melodic or accompanied sequences, or live arrangements of the grid, and he can easily and immediately switch phrases in real-time in a continuous musical discourse.

Finally, the part played by ImproteK can evolve during an improvisation session: it is actually able to be soloist and/or accompanist depending on the nature of the sequences chosen by the performer to fill the live oracle. Indeed, making this oracle listen to an accompanist's playing will lead to perform real-time arrangement of the harmonic grid. The second way to deal with accompaniment is the harmonization and arrangement module introduced in the following section.

5. HARMONIZED AND ARRANGED IMPROVISATIONS

The harmonization and arrangement module can be used in an autonomous way as an independent block producing an accompaniment for a melody without interacting in real-time. Yet, it was conceived to be integrated in the wider environment of ImproteK to compose new accompanied improvisations.

It does not use any musical rule and is exclusively based on a corpus which is simultaneously seen as a learning ground to extract empirical harmonization mechanisms, and as a musical memory in the framework of a pattern reuse technique to make the accompaniment concrete once it has been calculated.

5.1. Learning step

5.1.1. The corpus

The learning of the corpus is performed on three features: the *melodic track* (theme, solo improvisation, etc.), the *accompaniment track*, and the associated *harmonic grid*. It is carried out on live performances by musicians and consists for the moment in jazz standards and pieces by Bernard Lubat.

We refer to ”corpus” to designate the set of models extracted from an offline learning as well as those created during the current performance which are immediately available for generation: once again, the system listens to the musicians, segments the inputs by beat, and learns these sequences as well as the associations between the three features by building oracle objects. The learning step is actually double. A couple of oracles is built for every element of the corpus: an harmonization oracle, and an arrangement oracle (fig. 6).

The *harmonization oracle* records the sequence of associations between the melodic fragments and the chord labels of the beats it covers. In the case of the *arrangement*

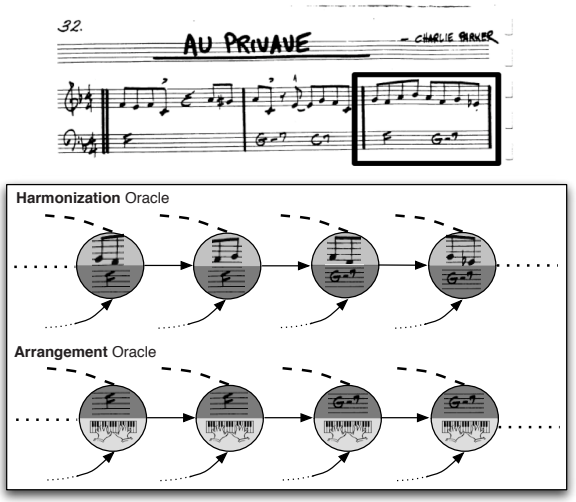

Figure 6. Learning the harmonization and arrangement oracles.

oracle, the concerned associations involve the chord labels and the fragments of accompaniment. As their names suggest, they will be used to associate a symbolic chord progression with a melodic track, and a accompaniment track with a symbolic chord progression, respectively.

5.1.2. Equivalence

This incremental building, as wells as the pattern matching described in the next paragraph (5.2.1), is actually performed on equivalence classes.

In the case of a harmonization oracle, two states are considered as equivalent if they are indexed by the same notes without taking into consideration their duration, order, or repetition in the beat slice. In the same way, two states of an arrangement oracle are equivalent if they are indexed by the same chord label whatever the associated fragment of accompaniment is.

These equivalences have been introduced to avoid the unproductive rigidity which could have been brought by a strict equality criterion. The consequence regarding the oracles' structure impacts the number of operating suffix links as illustrates the simplified example in fig 7, applied to a character string: an oracle built on the word *oracle*.

In the first example, every suffix link points on the initial state because no repeated sub-pattern has been found (every letter only appears once). In the second example using the equivalence classes *vowels* and *consonants*, we observe repeated sub-patterns and the structure is much more complex.

5.2. Generation step

5.2.1. Harmonizing and arranging in a cascade

The whole harmonization and arrangement process amounts to repeating the mechanism of constrained navigation in a cascade, first with a chosen harmonization oracle then with a chosen arrangement oracle (fig. 8).

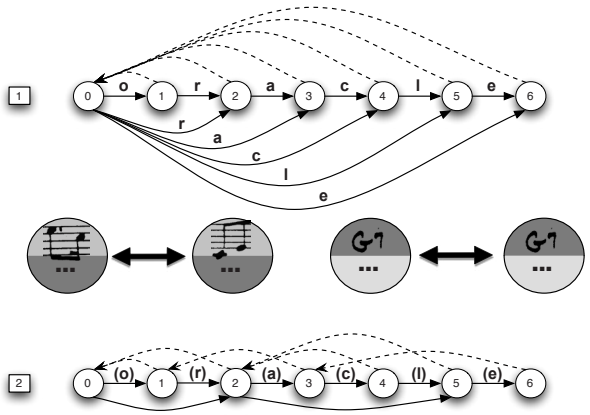

Figure 7. Suffix links and equivalence classes.

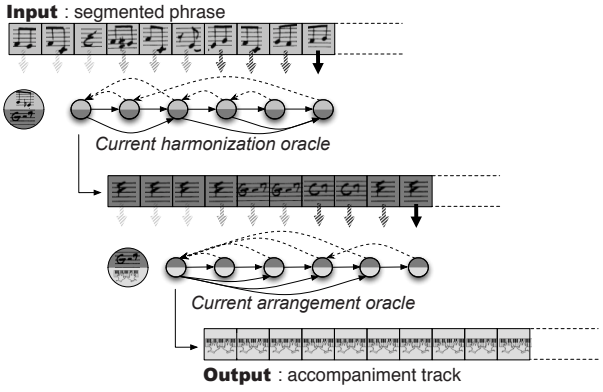

Figure 8. Harmonization and arrangement process.

The harmonization outputs a chord labels progression, and this symbolic sequence therefore becomes the path to follow for the navigation in the chosen arrangement oracle (filled by sequences of associations between melodic fragments and chord label sliced by beat). The pattern matching is this time performed on chord labels to output a sequence formed by the concatenation of the accompaniment fragments found in every stage of the research.

5.2.2. Formal intermediary

It is important to note that, even if an intermediate step involves symbolic data such as chord labels, no harmonic rule is used to perform harmonization and arrangement. The denomination of this formal intermediary is only user-oriented to make performance more intuitive: the system itself is unaware of the musical meaning of this labeling and only considers two chord labels as two indexes to compare.

The insertion of this formal language at an intermediate level separating the process in two different steps is motivated by three reasons. First, it naturally comes from the usual notation in jazz scores as we found in the Realbooks where a melody is facing a corresponding chords label progression. Then, it enables to multiply the possibilities: a phrase can indeed be harmonized with a given oracle and then arranged using an arrangement oracle learned on a completely different corpus. Finally, it will allow to implement in a future development an optional level of chord substitutions based on a grammar [10].

Furthermore, the terms "harmonization" and "arrangement" come from the fact that ImproteK has been used so far in tonal jazz sessions. In other musical contexts, its genericity enables an understanding of other forms of vertical associations that can be indexed in an agnostic way with an other grammar.

6. EXPERIMENTS AND RESULTS

ImproteK has been used as a virtual partner by professional musicians, in particular French jazz musician Bernard Lubat during improvisation sessions conducted in Uzeste in 2011. Video and audio examples can be found at *http://ehess.modelisationsavoirs.fr/improtech/improtek*.

Some of them show the real-time control of the software through the Max/MSP interface (see for instance the improvisation based on Erroll Garner's mambo style transcriptions). The direct interaction of the computer with a live musician is illustrated by improvisation sessions where Bernard Lubat and ImproteK alternately play as soloist and accompanist, or trade choruses.

Other series illustrate the wide variety of results for a same input in the harmonization and arrangement module depending on the user's choice to use different parts of the corpus for both steps, and on the continuity he imposed. It goes from "imitation" by choosing a same part of the corpus for the live, harmonization, and arrangement oracles, to originality or even extravagance when completely unrelated oracles are used.

7. CONCLUSION

We described a music generation system which is able to understand the logic of the horizontal and vertical associations in a live musical improvisation performance to become itself a source of proposals by developing its own aesthetics close to that of its partners. Its prime material is indeed their playing. It is used at the same time as a learning ground for the style modeling, and as a musical memory to develop its own improvisations.

The properties of the oracle structuring this memory enable to get over the dilemma "innovation vs. coherence" by ensuring continuity by construction, and therefore giving the possibility to work with a fine grain: the beat is set as the elementary unit in the calculation, and its restitution is made possible by following a beat tracker to reach a better interaction.

ImproteK is indeed conceived as a proper instrument and requires a full-time performer to manage the learning, the generation, and the playing in real-time. It can alternately be soloist or accompanist and is even capable of creating accompanied improvisations via the harmonization and arrangement module.

Current work is devoted to the evaluation of the compatibility between the harmonic progression of the current session and that of the accompaniment returned by this module. For the moment, the performer is given the comparison between both grids through the interface which displays the respective chord labels. This study will lead to a better integration of the "harmonic interaction" in the instrument, and will make its use more intuitive.

Acknowledgment

This work is realised with the support of the French National Research Agency, in the framework of the project "IMPROTECH", ANR-09-SSOC-068.

We wish to thank the OMax family Gérard Assayag, Georges Bloch, and Benjamin Lévy for the fruitful exchange of experiences and ideas regarding the conception and implementation of ImproteK. We thank Laurent Bonnasse-Gahot who made it get rhythm with the beat tracker, Carlos Agon and Jean Bresson for their advice concerning OpenMusic, and Arshia Cont for the custom-made Antescofos. Finally, we want to express special thanks to *La Compagnie Lubat* for the always enriching sessions.

8. REFERENCES

[1] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor oracle: A new structure for pattern matching," in *SOFSEM 99: Theory and Practice of Informatics*. Springer, 1999, pp. 758–758.

[2] G. Assayag and G. Bloch, "Navigating the oracle: A

heuristic approach," in *International Computer Music Conference*, vol. 7, 2007, pp. 405–412.

[3] G. Assayag, G. Bloch, and M. Chemillier, "Omax-ofon," *Sound and Music Computing (SMC)*, 2006.

[4] G. Assayag, G. Bloch, M. Chemillier, A. Cont, and S. Dubnov, "Omax brothers: a dynamic topology of agents for improvization learning," in *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*. ACM, 2006, pp. 125–132.

[5] G. Assayag and S. Dubnov, "Using factor oracles for machine improvisation," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 8, no. 9, pp. 604–610, 2004.

[6] G. Assayag, S. Dubnov, and O. Delerue, "Guessing the composers mind: Applying universal prediction to musical style," in *Proceedings of the International Computer Music Conference*, 1999, pp. 496–499.

[7] J. Biles, "Genjam: Evolutionary computation gets a gig," in *Proceedings of the 2002 Conference for Information Technology Curriculum, Rochester, New York, Society for Information Technology Education*, 2002.

[8] L. Bonnasse-Gahot, "Donner à omax le sens du rythme: vers une improvisation plus riche avec la machine," École des Hautes Études en sciences sociales, Tech. Rep., 2010, http://ehess.modelisationsavoirs.fr/improtech/docs/L.Bonnasse-Gahot-beat-tracking2010.pdf.

[9] J. Bresson, C. Agon, and G. Assayag, "Openmusic 5: A cross-platform release of the computer-assisted composition environment," in *10th Brazilian Symposium on Computer Music, Belo Horizonte, MG, Brésil*, 2005.

[10] M. Chemillier, "Toward a formal study of jazz chord sequences generated by steedman´ s grammar," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 8, no. 9, pp. 617–622, 2004.

[11] C. Chuan and E. Chew, "A hybrid system for automatic generation of style-specific accompaniment," in *4th Intl Joint Workshop on Computational Creativity*, 2007.

[12] A. Cont, "Antescofo: Anticipatory synchronization and control of interactive parameters in computer music," in *Proceedings of the International Computer Music Conference*, 2008.

[13] B. Lévy, "Visualising omax," Master's thesis, Master ATIAM, Université Pierre et Marie Curie, Paris VI - IRCAM, 2009.

[14] G. Lewis, "Too many notes: Computers, complexity and culture in voyager," *Leonardo Music Journal*, pp. 33–39, 2000.

[15] J. Nika, "Intégrer l'harmonie dans un processus informatique d'improvisation musicale," Master's thesis, Master ATIAM, Université Pierre et Marie Curie, Paris VI - IRCAM, 2011, http://articles.ircam.fr/textes/Nika11a/index.pdf.

[16] F. Pachet, "The continuator: Musical interaction with style," *Journal of New Music Research*, vol. 32, no. 3, pp. 333–341, 2003.

[17] A. Patel, J. Iversen, M. Bregman, I. Schulz, and C. Schulz, "Investigating the human-specificity of synchronization to music," in *Proceedings of the 10th International Conference on Music and Cognition. Sapporo, Japan*, 2008, pp. 100–104.

[18] G. Ramalho, P. Rolland, and J. Ganascia, "An artificially intelligent jazz performer," *Journal of New Music Research*, vol. 28, no. 2, pp. 105–129, 1999.

[19] R. Rowe, *Interactive music systems: machine listening and composing*. MIT press, 1992.

[20] I. Simon, D. Morris, and S. Basu, "MySong: automatic accompaniment generation for vocal melodies," in *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. ACM, 2008, pp. 725–734.

[21] B. Thom, "BoB : an interactive improvisational music companion," in *Proceedings of the fourth international conference on Autonomous agents*. Citeseer, 2000, pp. 309–316.

[22] D. Zicarelli, "M and jam factory," *Computer Music Journal*, vol. 11, no. 4, pp. 13–29, 1987.