HAL
open science

Omax Brothers : a Dynamic Topology of Agents for Improvization Learning

Gérard Assayag, Georges Bloch, Marc Chemillier, Arshia Cont, Shlomo Dubnov

To cite this version:

Gérard Assayag, Georges Bloch, Marc Chemillier, Arshia Cont, Shlomo Dubnov. Omax Brothers : a Dynamic Topology of Agents for Improvization Learning. Workshop on Audio and Music Computing for Multimedia, ACM Multimedia 2006, Oct 2006, Santa Barbara, United States. pp.1-1. hal-01161351

HAL Id: hal-01161351

https://hal.science/hal-01161351v1

Submitted on 8 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

OMax Brothers : a Dynamic Topology of Agents for Improvization Learning

Gérard Assayag
IRCAM-CNRS UMR Stms
1 place I. Stravinsky
75004 Paris, France
+33 1 44 78 48 58
assayag@ircam.fr

Georges Bloch
University of Strasbourg
22 rue René Descartes
67084 Strasbourg France
+33 1 03 88 41 73 54
gbloch@umb.u-strasbg.fr

Marc Chemillier
University of Caen
Bd Maréchal Juin
BP 5186F-14032 Caen Cedex
+33 (0)2 31 45 25 04
chemilli@free.fr

Arshia Cont
IRCAM-UCSD
1 place I. Stravinsky
75004 Paris, France
+33 1 44 78 48 58
arshia.cont@ircam.fr

Shlomo Dubnov
UCSD
9500 Gilman Dr. MC 0326
La Jolla, CA 92093-0326
(858) 534-3230
sdubnov@ucsd.edu

ABSTRACT

We describe a multi-agent architecture for an improvization oriented musician-machine interaction system that learns in real time from human performers. The improvization kernel is based on sequence modeling and statistical learning. The working system involves a hybrid architecture using two popular composition/perfomance environments, Max and OpenMusic, that are put to work and communicate together, each one handling the process at a different time/memory scale. The system is capable of processing real-time audio/video as well as MIDI. After discussing the general cognitive background of improvization practices, the statistical modeling tools and the concurrent agent architecture are presented. Finally, a prospective Reinforcement Learning scheme for enhancing the system's realism is described.

Categories and Subject Descriptors

J.5 [**Computer Applications**]: Art and Humanities – performing arts (e.g. music).

General Terms

Algorithms, Design, Experimentation, Human Factors.

Keywords

Improvization, machine learning, variable memory markov systems, concurrent agents, sequence modeling, style modeling, statistical learning, computer music, man machine interaction.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AMCMM'06, October 27, 2006, Santa Barbara, California, USA.
Copyright 2006 ACM 1-59593-500-2/06/0010...$5.00.

1. INTRODUCTION

Machine improvization and related style learning problems usually consider building representations of time-based media data, such as music or dance, either by explicit coding of rules or applying machine learning methods. Stylistic machines for dance emulation try to capture movement by training statistical models from a sequence of motion-capture sequences [6]. Stylistic learning of musical style use statistical models of melodies or polyphonies to recreate variants of musical examples [8]. These and additional research indicate that emulation of particular behaviors is possible and that credible behavior could be produced by a computer for a specific domain.

In the field of music improvization with computers there has been recently notable advances in statistical music modeling that allows capturing stylistic musical surface rules in a manner that allows musically meaningful interaction between humans and computers. We have experimented with several of these models during the past years, and more recently implemented OMax, an environment which benefits both from the power of (Lisp based) OpenMusic [4] for modeling and high level programming, and MaxMSP [15] for performance and real time processing. This environment allows for interaction with one or more human player, on-the-fly stylistic learning, virtual improvization generation, metrical and harmonic alignment, stylistic model archiving and hybridation. It operates on two distinct time scales : the Max one, which is close to real time and involves fast decision/reaction, and the OpenMusic one, which has a deeper analysis/prediction span over the past and the future. These two conceptions of time interact and synchronize over communication channels through which musical data as well as control signals circulate in both directions. A decisive advantage we have found in this hybrid environment experience is its double-folded extendability. In the OM domain, it is easy, even while the system is running, to change the body of a lisp function and test incremental changes. Furthermore, it is easy to enrich the generation by connecting the system to a wide variety of compositional algorithms available in this environment. Same thing in the Max domain, with a comprehensive collection of

real-time generation and processing modules. We think of this setup more as an indefinitely modulable and extendible experimental environment for testing new ideas about interaction, than as a fixed application.

2. INTERACTION, IMPROVIZATION AND LEARNING

The aim of the early interactive computer pieces, as theorized in the late seventies by Joel Chadabe, was of course to create a consistent musical style, adaptive to the performer. However, when this kind of piece was transported to a real improvization setup, the goal changed to "composed improvization" [5]. And the consistency had to be as much a consistency of style per se as a consistency with the style of the improvizer. One had to recognize the performer throughout the system. The first software to achieve this stricter kind of consistency is probably *M* by Chadabe and Zicarelli [24]. This early macintosh MIDI-based environment would "listen" to a musician's performance and build on the fly a markov chain representation of the MIDI stream, then walks through this representation in order to send a musical feed-back

The interesting thing about M was its ability to send back a stylistically consistent mirror image of the performer to the performer, which would, as an answer to this feed-back, change his way of playing accordingly. We call this process stylistic reinjection, and a similar idea has been explored by François Pachet under the name of "reflexive interaction" [25][26].

Another interesting case is the GenJam system by John A. Biles. GenJam implements a genetic algorithm that grows a population of musical phrase-like units in a highly supervised mode. The well fitted phrases are played in an interactive manner as a response to the musician in a traditional jazz "trading fours" dialog.

Other remarkable environments such as the Voyager system by George Lewis [18], the Ramses system by Steve Coleman, or the experiments carried on by David Wessel using Don Buchla digital instruments and sophisticated parameter mapping programmed in Max deserve great attention [20], but are outside the scope of this paper as they do not use machine learning schemes, and should be considered more related to algorithmic music generation.

We will focus here on the question of providing a virtual musical partner that learns all its knowledge from the musicians it's playing with, in a non supervised mode, and that is fitted to a real-time audio context.

The musical hypothesis behind stylistic reinjection is that an improvizing performer is informed continually by several sources, some of them involved in a complex feed-back loop (see Figure 1). The performer listens to his partners. He also listens to himself while he's playing, and the instantaneaous judgement he bears upon what he is doing interferes with the initial planning in a way that could change the plan itself and open up new directions. Sound images of his present performance and of those by other performers are memorized, thus drifting back from present to the past. From the long term memory they also act as inspiring sources of material that will eventually be recombined to form new improvized patterns. We believe that musical patterns are not stored in memory as literal chains, but rather as compressed models, that may, upon reactivation develop into similar but not identical sequences : this is one of the major issues behind the balance of recurrence and innovation that makes an interesting improvization. The idea behind stylistic reinjection is to reify, using the computer as an external memory, this process of reinjecting musical figures from the past in a recombined fashion, providing an always similar but always innovative reconstruction of the past. To that extent, the virtual partner will look familiar as well as challenging.

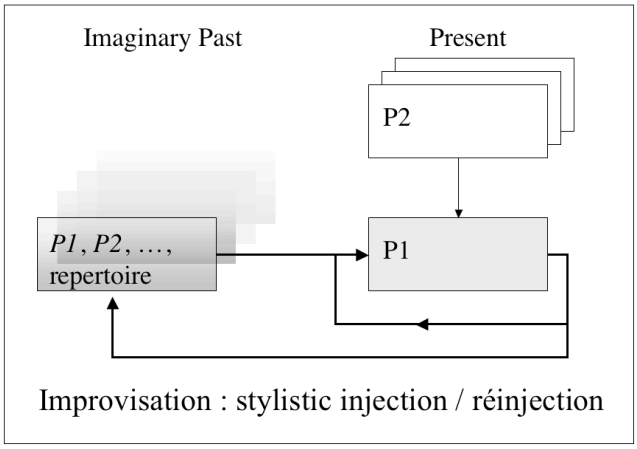

Figure 1. Stylistic Reinjection

3. STATISTICAL MODELING

Statistical modeling of musical sequences has been the subject of experimentation since the very beginning of musical informatics [8]. The idea behind *context models* we use, is that events in a musical piece can be predicted from the sequence of preceding events. The operational property of such models is to provide the conditional probability distribution over an alphabet given a preceding sequence called a context. This distribution will be used for generating new sequences or for computing the probability of a given one. First experiments in context based modeling made intensive use of Markov chains, based on an idea that dates back to Shannon : complex sequences do not have an obvious underlying source, however, they exhibit a property called *short memory property* by Ron et al [27]; there exists a certain memory lengh L such that the conditional probability distribution on the next symbol does not change significantly if we condition it on *contexts* longer than L. In the case of Markov chains, L is the order. However, the size of Markov chains is $O(|\Sigma|^L)$, so only low order models have been actually implemented.

To cope with the model order problem, in earlier works [3, 10-13] we have proposed a method for building musical style analyzers and generators based on several algorithms for prediction of discrete sequences using Variable Markov Models (VMM). The class of these algorithms is large and we focused mainly on two variants of predictors - universal prediction based on Incremental Parsing (IP) and prediction based on Probabilistic Suffix Trees (PST).

From these early experiences we have drawn a series of prescriptions for an interactive music learning and generating method. In the following, we consider a learning algorithm, that builds the statistical model from musical samples, and a generation algorithm, that walks through the model and generates a musical stream by predicting at each step the next musical unit from the already generated sequence. These prescription could be summed up as :

- learning must be incremental and fast in order to be compatible with real-time interaction, and be able to

switch instantly to generation (real-time alternation of learning and generating can be seen as « machine improvization » where the machine « reacts » to other musicians playing).

- The generation of each musical unit must be bounded in time for compatibility with a real time scheduler.
- In order to cope with the variety of musical sources, it is interesting to be able to maintain several models and switch between them at generation time.
- Assuming the parametric complexity of music (multi-dimensionality and multi-scale structures) multi-attribute models must be searched, or at least a mechanism must be provided for handling polyphony.

We have chosen for OMax a model named *factor oracle* (FO) that conforms with points 1, 2 and 4. It is described in detail [1] and its application to music data is described in [2]. An example is given in Figure 2.

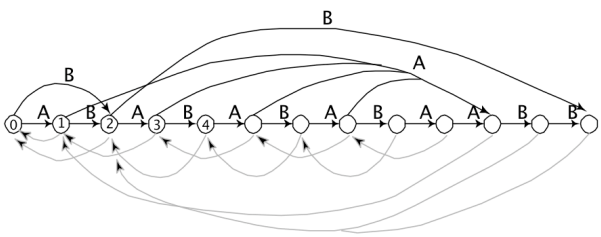

Figure 2. A factor oracle for the string ABABABABAABB.

FO's capture all sub-phrases (factors) in a sequence of symbols, transformed into a linear chain of states by an efficient incremental algorithm. Through this transformation, the sequence structure is "learned" in the FO. The states are linked by two kind of arrows. Forward arrrows are called *factor links*. By following these at generation time, it is possible to generate factors of the original sequence, i.e. literally repeat learned subsequences. Backward arrows are called suffix links. By following these, it is possible to switch to another subsequence sharing a common suffix with the current position. Such a recombination, is really a *context* based generation, the context being the common suffix. Although the probability model has not yet been defined, FO's are conjectured to be VMM's.

Basically a FO is a finite state automaton constructed in linear time and space in an incremental fashion. A sequence of symbols $s = s_1 s_2 .. s_n$ is learned in such an automaton, the states of which are S_0, S_1, S_2 .. S_n. There is always a transition arrow (called factor link) labelled by symbol s_i going from state S_{i-1} to state S_i, $1 \leq i < n$. Depending on the structure of s, other arrows will be added to the automaton. Some are directed from a state S_i to a state S_j, where $0 \leq i < j \leq n$. These also belong to the set of factor links and are labelled by symbol s_j. Some are directed « backward », going from a state S_i to a state S_j, where $0 \leq j < i \leq n$. They are called suffix links, and bear no label. The factor links model a factor automaton, that is every factor p in s corresponds to a unique factor link path labeled by p, starting in S_0 and ending in some other state. Suffix links have an important property : a suffix link goes from S_i to S_j iff the longest repeated suffix of s[1..i] is recognized in S_j. Thus suffix links connect repeated patterns of s.

The oracle is learned on-line. For each new entering symbol si, a new state Si is added and an arrow from S_{i-1} to S_i is created with label s_i. Starting from S_{i-1}, the suffix links are iteratively followed backward, until a state is reached where a factor link with label si originates (going to some state S_j), or until there is no more suffix link to follow. For each state met during this iteration, a new factor link labeled by si is added from this state to S_i. Finally, a suffix link is added from S_i to the state S_j or to state 0 depending on which condition terminated the iteration.

Navigating the oracle in order to generate variants is straightforward : starting in any place, following factor links generates a sequence of labelling symbols that are re-iterations of portions of the learned sequence ; following one suffix link followed by a factor links creates a recombined pattern sharing a common suffix with an existing pattern in the original sequence. This common suffix is, in effect, equivalent to the context in the context-inferences model. In addition to completeness and incrementality of this model, the best suffix is known at the minimal cost of just following one pointer. By following more than one suffix link before going back to the factor generation, or by reducing the number of successive factor link steps, we make the generated variant less similar to the original.

FO's are functionally close to suffix trees, but with much fewer nodes. In comparison to IP and PST trees that discard substrings, FO's are preferred because they can be built quickly and, like the suffix tree, they encode all possible substrings. One of the main properties of FO's is that they index the input sequence in such a way that, at every point along the data, they constructs pointers to possible continuations for most recent suffixes at that point. By "recent suffixes" we mean suffixes that occur for the first time when a new symbol is observed. Since FO's are constructed online, all "previously seen" suffixes are detected earlier in the sequence. So, at every point along the sequence FO's provides pointers to continuations for most recent suffixes, and a pointer back to the longest repeating suffix. F0 is used to generate new improvization by either jumping into the "future" based on the most recent past, or by going back to the more distant past to look for continuations of previously encountered suffixes. So, instead of considering the best context with log-loss "gambling" on the next note, the FO method operates by "forgetting" and selective choice of historical precedence for deciding the future.

4. OMax BROS ARCHITECTURE

OMax is distributed across two computer music environments : OpenMusic and Max (see Figure 3). Obviously, the Max components are dedicated to real-time interaction, instrumental capture, MIDI and audio control, while OpenMusic components are specialized in higher level operations such as building and browsing the statistical model. Communication channels between Max and OM allow the transmission of streams of musical data as well as control information. In the primitive version of OMax, Midishare [14] has been used as an inter-application communication system, but all the transmitted data had to be encoded in MIDI. Of course this is not very convenient, specially when structured data has to be exchanged, so this component has been rewritten using OSC [17], a powerful message based communication protocol that is increasing in popularity in the computer music community.

At the input and the output, Max handles the direct interaction with the performers. It extracts high level features from the sound signal such as the pitch, velocity, onset-offset, and streams them to OpenMusic using the OSC protocol. The data flowing in this channel is called "augmented MIDI" because it contains MIDI-like symbolic information, plus any necessary relevant information regarding the original signal. OpenMusic

builds up incrementally the high level representations derived from the learning process. Simultaneously, it generates an improvization from the learned model and outputs it as an "augmented MIDI" stream. At the output, Max reconstructs a signal by taking advantage of the augmented data. For example, the signal feature could be the pitch as extracted by a pitch tracker. The augmented information could be pointers into the recorded sound buffer, mapping the MIDI information to sound events. The reconstruction process could be concatenative synthesis that would mount in real-time the sound units into a continuous signal.

Of course, the input could be restricted to MIDI, and the output could be restricted, in any case, to controlling some expander using the MIDI data and ignoring the augmented data, or any such combination. We develop here the most difficult case which is going from audio to audio.

Right now we have mostly experimented on pitch extraction, but of course any descriptor could be used, including spectral or gesture descriptors, and any reconstruction could be tried, including pure synthesis.

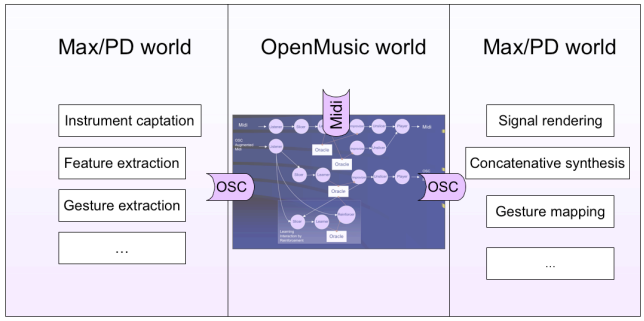

Figure 3. OMax architecture

Inside the OpenMusic world lives a community of concurrent agents that can be freely instantiated and arranged into different topologies. These agents belong to six main classes :

- Listener
- Slicer
- Learner
- Improvizer
- Unslicer
- Player

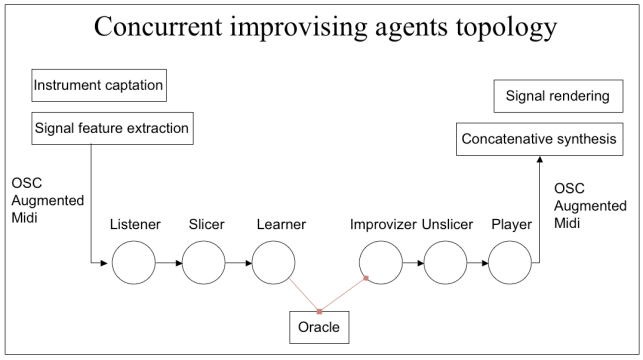

Figure 4. A simple agent topology

Figure 4 shows a typical topology where the augmented MIDI stream is prepared into some convenient form by the listener and slicer agents and provided to a learner process that feeds the Oracle structure. Concurrently, an Improvizer process walks the oracle and generate a stream that is prepared in order to feed the rendering engine in Max.

The statistical modeling techniques we use suppose the data to be in the form of sequences of symbol taken from an alphabet. Of course, because music is multidimensional, it has to be processed in some way in order for the model to be usable and meaningful. We detail in [11] a method for processing polyphonic streams in order to turn them into a sequence of symbols such that a sequence model can be built from which new generated strings can be easily turned into a polyphony similar in structure to the original. Such "super-symbols", output by the polyphony manager in OM, are "musical slices" associated with a certain pitch content and a duration. Two processes, the "slicer" and the "unslicer" will be dedicated to transforming the raw augmented MIDI stream into the slice representation used by the model, then back into a polyphonic stream (see Figure 5).

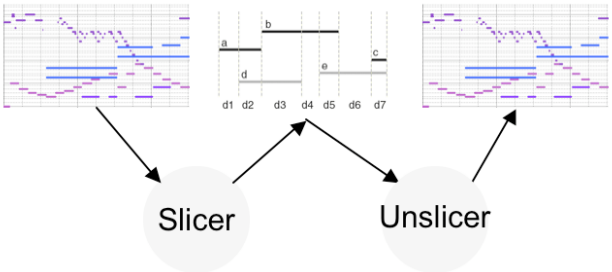

Figure 5. Slicing polyphony

OMax provides a toolkit for easily setting up different agent topologies in order to experiment with a variety of musical situations.

4.1 Example 1 : a simple OMax topology

For example, in Figure 6, two musical sources (MIDI or audio) are merged into the same slicer, which means that the sliced representation at the output of the process will account for the overall polyphony of the two sources. In such a case, the Oracle learns not only the pattern logic of both musicians, but also the way they interact. Improvizing on such an oracle will generate a polyphonic stream that respects the vertical as well as horizontal relations in the learned material. The learner process here feeds three different Oracles. Such a configuration may prove useful either for splitting the musical information into different points of view (e.g. pitch versus rhythm) or in order to learn different parts of the performance into different Oracles so they are not polluted one by the other. Then the three Oracles are improvized by three independent Improvizers, either simultaneously or in turn. Many interesting comparable topologies can be tested. The agent connectivity is implemented using (invisible) connection agents that provide dynamic port allocation so the program can instantiate communication ports and connect agent input and output.

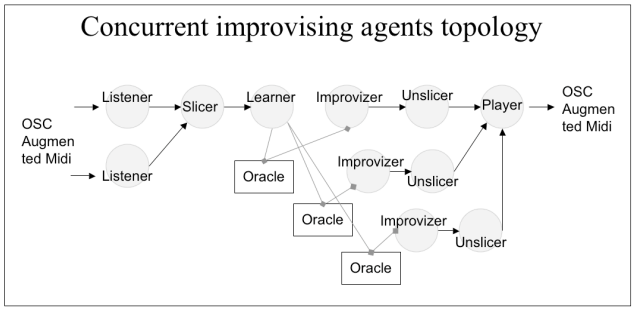

Figure 6. Another agent topology.

4.2 Example 2 : a meta-learning topology

This configurable agent topology is ready for experiments that go well beyond the machine improvization state of the art, by adding a meta-learning level. In Figure 7, the agents in the rectangle at the bottom learn in a separate oracle a "polyphony" made up from the musical information issued by the listener, and from the *states* of the first-level oracle just above the rectangle. What the bottom oracle learns really is the correlation between what is played by the musician and by the oracle simultaneously, that is it learns the *interaction* between them (considering that the musician always adapts his performance with regard to what the oracle produces). Learning the *states* instead of the *output* of the oracle means that if the human tries later to reinstall the same musical situation, the system is then able, through a new module called a reinforcer, to get back to the original oracle in order to reinforce the learned states. The effect of this architecture would be a better musical control of the global form by the human performer, and the feeling that OMax *understands* actually the message that is implicitly sent by a musician when he gets back to a previously encountered musical situation : the message in effect could be that he liked the interaction that occurred at that time and would like the computer to behave in a similar way.

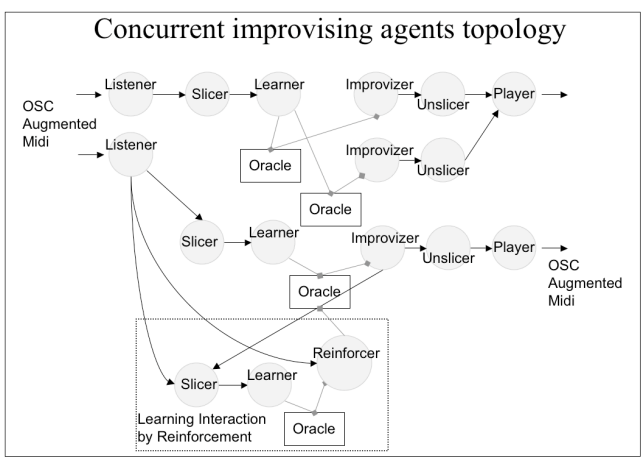

Figure 7. An agent topology with a meta-oracle.

5. REINFORCEMENT LEARNING

One of the common methods to simulate anticipatory planning has been the use of reinforcement learning techniques with reward signals calculated on expected future states [21].

In a RL (Reinforcement Learning) system, rewards are defined for goal-oriented interaction. In musical applications, defining a goal would be either impossible or would limit the utility of the system to certain styles. In this sense, the rewards used in our interaction are rather guides to evoke or repress parts of the learned model in the memory. We define two execution modes for our system demonstrated in Figure 8.

In the first, referred to as Interaction mode, the system is interacting with a human performer. During the second mode, called self listening mode, the system is in the generation phase and is interacting with itself, or in other words it is listening to itself, in order to decide how to proceed.

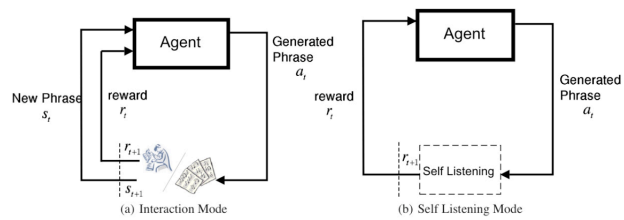

Figure 8. The Reinforcement Learning scheme

The agent in both modes consists of a model-based RL framework. It consists of an internal model of its environment and reinforcement learning for planning. This internal model plays the role of memory and representation of new inputs. The main feature of the agent is that it handles different types of musical representations (or viewpoints) as separate models. The RL algorithm used would then be a one with collaborative and competitive learning between viewpoints.

The interaction mode occurs when external information is being passed to the system from the environment (human improvizer). This way the reward would be the manner in which this new information reinforces (positively or negatively) the current stored model in the agent. The self listening mode occurs when the system is improvising new sequences. In this mode, the RL framework would be in a model-free learning state, meaning that the internal model of the environment stays intact but the planning is influenced by the latest musical sequence that has been generated by the system itself, thus, the idea of self listening.

When a new sequence S^t with length N is received from the environment, an ideal reward signal should reinforce the part of memory which most likely evoke the sequence received to be able to generate recombinations or musically meaningful sequences thereafter. In the RL framework, this means that we want to assign numerical rewards to transition states and suffix states of an existing FO. Reward computation occurs before integrating the new sequence into the model.

After different attributes of S^t are extracted as separate sequence (each in the form $\{x_1 \ldots x_N\}$), we use a probability assignment function P from $S^* \rightarrow [0, 1]$ (where S^* is the set of all available tuple of states) to assign rewards to states in the model as follows:

$$P(s_{1^*} s_{2^*} \ldots s_{N^*} | S^t) = \left[\sum_{i=1}^{N} p(x_i | s_i^*) \right] / N$$

where

$$p(x_i, s_i) \quad = \quad \begin{cases} 1 & \text{if } F_{trn}(s_{i-1}, x_i) = s_i \\ 0 & \text{if } F_{trn}(s_{i-1}, x_i) = \emptyset \end{cases}$$

and F_{trn} is the factor link between two states.

The meaning of this equation is that it reinforces the states in the memory that can most regenerate all of the original sequence. In other words, it will guide the learning described later to the places in the memory that should be most regarded during learning and generation. For example, for an attribute sequence $\{e_1, \ldots, e_N\}$, a sequence of states $\{s_1^*, \ldots, s_N^*\}$ in a FO would get a reward of 1 if it regenerates exactly the original S^t.

To assign rewards to suffix links, we recall that they refer to previous states with the largest common suffix. Using this knowledge, a natural reward for a suffix link would be proportional to the length of the common suffix that the link is referring to. Fortunately, using FO's, this measure can be easily calculated online. It has been introduced in [22], and is adopted here as well.

Rewards or guides are calculated the same way for both modes of the system described before with a small difference. We argue that the rewards for the interaction mode correspond to a psychological attention towards some part of the memory and guides for the self-listening mode correspond to a preventive anticipation scheme. This means that during interaction with a human improvisor, the system needs to be attentive to (new) input sequences from the environment and during self-listening in needs to be preventive so that it would not generate the same (optimal) path over and over. For this reason, rewards calculated using the above formula will get a negative sign for the self-listening mode.

In order to implement full RL, rewards are not enough. A *policy* has to be defined. A policy is a strategy for choosing the next action with regard to the current environment. In our case, a policy will be a probability weighting of possible transitions in the FO, considering not only the current rewards but also the estimation of how these rewards will progress in the future. In effect, considering only the current rewards would lead to poor planning, as a highly valued path in the FO could be attained through less valued transitions.

In order to achieve this, a policy matrix is maintained with the FO states as lines and the music vocabulary (called « actions » in the RL terminology) as columns. After the rewards update has been performed, a policy update is done by « practicing », i.e. generating silently a certain number of improvizations paths of a fixed length. For each consecutive transition (a state-action pair) we update the corresponding cell in the policy matrix by accumulating the reward associated with the transition. This is summarized in the following formula :

$$Q(s_m, a_m) = Q(s_m, a_m) + \alpha \left[R(s_m) + \gamma \cdot \max_{a'}(Q(s_m, a')) - Q(s_m, a_m) \right]$$

$$R(s_t) = \sum r(s_t) + \gamma r(s_{t+1}) + \cdots + \gamma^m r(s_{t+m}) + \cdots$$

The Policy Matrix Q is updated at position s_m, a_m. $R(s_m)$ is the integration of the reward value for state s_m over an infinite horizon. The γ coefficient, called discount factor, ensures that this calculation finishes as γ^m decreases down to ε.The sequence of successive states/rewards is obtained by following greedily a chain of states in Q, just as it would be done for effective generation, starting at s_m and choosing the best policy. α is a learning rate that controls the effectiveness of the update.

At generation time, the FO transitions will be followed as usual, but the policy will arbitrate the choices between concurrent transitions by checking the values in Q.

6. CURRENT STATE / PERSPECTIVES

The current version of OMax is implemented in OpenMusic and Max. The OpenMusic version provides a general framework for musical agents which frees the developer from the details of handling agent messaging, inter-agent agent data streaming, and agent synchronization. This framework has been implemented using meta-programming techniques so that an agent instance comes automatically equipped with all the concurrency management gear, letting the developer program using the usual CLOS (Common Lisp Object System) abstractions (classes, methods, slots, etc.). This framework is inspired by NetClos, a package for distributed concurrent programming in Lisp [23].

On the Max side, the audio stage uses intensively the Yin~ pitch follower object designed by Alain de Cheveigné and implemented by Norbert Schnell [9]. We have developed a yin post processor that increases significantly the pitch detection quality and provides a valuable onset-offset detection.

The RL scheme presented in the last section is still a Matlab prototype, which we have to optimize for real-time. The RL techniques will be experimented in two directions : increasing the intrinsic quality of the generation, and learning the large scale planning of the human performer and the way he reacts to the machine improvization, by the use of a meta-oracle that learns the interaction.

Examples (audio, midi and video) can be found at :

http://recherche.ircam.fr/equipes/repmus/OMax/

The video examples have been realized using the sound desriptors as a simple means of segmenting and recombining the images. This aspect of OMax has not been detailed in this paper and is left for future presentation.

7. REFERENCES

[1] Allauzen C., Crochemore M., Raffinot M., Factor oracle: a new structure for pattern matching, *Proceedings of SOFSEM'99, Theory and Practice of Informatics, J. Pavelka, G. Tel and M. Bartosek ed., Milovy, Lecture Notes in Computer Science 1725*, pp 291-306, Springer-Verlag, Berlin, 1999.

[2] Assayag, G., Dubnov, S., Using Factor Oracles for Machine Improvization, G. *Assayag, V. Cafagna, M. Chemillier (eds.), Formal Systems and Music special issue, Soft Computing 8*, pp. 1432-7643, September 2004.

[3] Assayag, G., Dubnov, S., Delerue, O., "Guessing the Composer's Mind: Applying Universal Prediction to Musical Style," *Proc. Int'l Computer Music Conf., Int'l Computer Music Assoc.,* pp. 496-499, 1999.

[4] Assayag A., Rueda C., Laurson, M., Agon C., Delerue, O. "Computer Assisted Composition at Ircam: PatchWork and OpenMusic," *The Computer Music J., vol. 23, no. 3,* 1999, pp. 59-72.

[5] Bloch, G., Chabot X., Dannenberg, R., «A Workstation in Live Performance: Composed Improvization», *Proceedings of International Computer Music Conference*, The Hague, Netherlands, 1986.

[6] Brand, M., and Hertzmann, A, 2000, Style Machines, *In Proceedings of SIGGRAPH 2000*, New Orleans, Louisiana, USA

[7] Chemillier, M., "Toward a formal study of jazz chord sequences generated by Steedman's grammar," *G. Assayag, V. Cafagna, M. Chemillier (eds.), Formal Systems and Music special issue, Soft Computing 8,* pp. 617-622, 2004

[8] Conklin, D. « Music Generation from Statistical Models », *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences, Aberystwyth, Wales, 30– 35, 2003.*

[9] de Cheveigné, A., Kawahara, H. "YIN, a fundamental frequency estimator for speech and music", *J. Acoust. Soc. Am. 111, 1917-1930, 2002.*

[10] Dubnov, S. Assayag, G., "Improvization Planning and Jam Session Design using concepts of Sequence Variation and Flow Experience", *Proceedings of Sound and Music Computing '05,* Salerno, Italy, 2005.

[11] Dubnov, S., Assayag, G., Lartillot, O., Bejerano, G., "Using Machine-Learning Methods for Musical Style Modeling," *IEEE Computer, Vol. 10, n° 38*, p.73-80, October 2003.

[12] Dubnov, S., Assayag, G. « Universal Prediction Applied to Stylistic Music Generation » *in Mathematics and Music, A Diderot Mathematical Forum*, Assayag, G.; Feichtinger, H.G.; Rodrigues, J.F. (Eds.), pp.147-160, Springer-Verlag, Berlin, 2002.

[13] Dubnov, S., Assayag, G., El-Yaniv, R. "Universal Classification Applied to Musical Sequences," *Proc. Int'l Computer Music Conf., Int'l Computer Music Assoc*., 1998, pp. 332-340.

[14] Orlarey, Y., Lequay " MidiShare : a Real Time multi-tasks software module for MIDI applications", *Proceedings of the International Computer Music Conference 1989*, Computer Music Association, San Francisco, 1989.

[15] Puckette, M. "Combining Event and Signal Processing in the MAX Graphical Programming Environment." *Computer Music Journal 15(3): 68-77, 1991.*

[16] Warusfel, O., Misdariis N., "Sound Source Radiation Synthesis: From Stage Performance to Domestic Rendering" in *Proceedings of 116th AES Convention, 2004*.

[17] Wright M., Freed, A., Momeni A.: "OpenSound Control: State of the Art 2003.", *Proceedings of NIME 2003: 153-159, 2003.*

[18] Roads, Curtis. 1985b. "Improvizations with George Lewis." In C. Roads, ed., *Composers and the Computer.* California : William Kaufmann. pp.76-87.

[19] Biles, John A., GenJam: Evolution of a Jazz Improviser. In P. J. Bentley and D. W. Corne (ed.), *Creative Evolutionary Systems*. San Francisco: Morgan Kaufmann, 2001.

[20] Wessel, D., Wright, M., New Interfaces For Musical Expression, *Proceedings of the 2001 conference on New interfaces for musical expression*, Seattle, Washington, 2001.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, MIT Press 1998.

[22] A. Lefebvre and T. Lecroq. Computing repeated factors with a factor oracle. In L. Brankovic and J. Ryan, editors, *Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms*, pages 145–158, Hunter Valley, Australia, 2000.

[23] Hotz, L., Trowe, M., NetCLOS - Parallel Programming in Common Lisp. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, PDPTA 1999, Las Vegas, Nevada,USA

[24] Zicarelli, D. (1987). M and Jam Factory. *Computer Music Journal*, 11(4): 13, 1987.

[25] Pachet F. *Interactions Réflexives*, Le feed-back dans la création musicale, *Rencontres musicales pluridisciplinaires,* GRAME, Lyon, 2006. http://www.grame.fr/RMPD/RMPD2006/

[26] Assayag G., Bloch, G., Chemillier, M., *Improvisation et réinjection stylistique, Rencontres musicales pluridisciplinaires*, GRAME, Lyon, 2006. http://www.grame.fr/RMPD/RMPD2006/

[27] Ron, D., Singer, Y., Tishby, N., "The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length," *Machine Learning*, vol. 25, 1996, pp. 117-149.