Interaction with Machine Improvisation

Gérard Assayag, George Bloch, Arshia Cont, Shlomo Dubnov

Interaction with Machine Improvisation

Gérard Assayag[1], George Bloch[2], Arshia Cont[1,3], and Shlomo Dubnov[3] *

[1] Ircam-CNRS UMR 9912, 1 Place Igor Stravinsky, 75004 Paris France.
{assayag,cont}@ircam.fr
[2] University of Strasbourg, 22 rue René Descartes, 67084, Stransbourg, France.
gbloch@umb.u-strasbg.fr
[3] Music Department, University of California in San Diego.
sdubnov@ucsd.edu

Abstract. We describe two multi-agent architectures for an improvisation oriented musician-machine interaction systems that learn in real time from human performers. The improvisation kernel is based on sequence modeling and statistical learning. We present two frameworks of interaction with this kernel. In the first, the stylistic interaction is guided by a human operator in front of an interactive computer environment. In the second framework, the stylistic interaction is delegated to machine intelligence and therefore, knowledge propagation and decision are taken care of by the computer alone. The first framework involves a hybrid architecture using two popular composition/performance environments, Max and OpenMusic, that are put to work and communicate together, each one handling the process at a different time/memory scale. The second framework shares the same representational schemes with the first but uses an *Active Learning* architecture based on collaborative, competitive and memory-based learning to handle stylistic interactions. Both systems are capable of processing real-time audio/video as well as MIDI. After discussing the general cognitive background of improvisation practices, the statistical modelling tools and the concurrent agent architecture are presented. Then, an Active Learning scheme is described and considered in terms of using different improvisation regimes for improvisation planning. Finally, we provide more details about the different system implementations and describe several performances with the system.

1 Interaction, Improvisation and Learning

The aim of the early interactive computer pieces, as theorized in the late seventies by Joel Chadabe, was to create a consistent musical style adaptive to the performer. However, when this kind of piece was transported to a real improvisation setup, the goal changed to *composed improvisation* [1]. In this conception, the consistency had to be as much a consistency of style per se as to a consistency with the style of the improviser. One had to recognize the performer throughout the system. The first software to achieve this stricter kind of consistency

* Authors in alphabetic order.

is probably `M` by Chadabe and Zicarelli [2]. This early Macintosh MIDI-based environment would *listen* to a musician's performance and build a Markov chain representation of the MIDI stream on the fly, then walk through this representation in order to send a musical feed-back. Other remarkable environments such as the *Voyager* system by George Lewis [3], the *Ramses* system by Steve Coleman, or the experiments carried on by David Wessel [4] using Don Buchla's digital instruments and sophisticated parameter mapping will not be considered here since they do not use machine learning schemes for capturing the style of the performer, and accordingly should be considered under the realms of algorithmic music generation. Another interesting case is the *GenJam* system by John A. Biles [5]. GenJam implements a genetic algorithm that grows a population of musical phrase-like units in a highly supervised mode. The high fitness phrases are played in an interactive manner as a response to the musician in a traditional jazz "trading fours" dialog.

The interesting thing about `M` was its ability to send back a stylistically consistent "mirror image" of the performer to herself, which would accordingly as an answer to this feed-back, change her way of playing. We call this process stylistic reinjection. François Pachet has explored a related idea under the title *reflexive interaction* [6, 7]. Accordingly, we will focus here on the question of interacting with style models via a virtual musical partner that operates in two modes: (1) it can learn to imitate the musician's playing style in a non supervised manner, and (2) it listens to musician's improvisation and tries to reinforce selection of those musical materials in its repertoire that best fit the jamming session.

This distinction between the two modes hints upon two different musical modalities or cognitive functions that are simulated in the system. In the first mode, the musician's interaction with musical phrases and style can be considered as a sort of selective musical memory where stylistically important relations and novel possibilities of recombination are recorded. This memory structure, if used for generation in an unsupervised manner, produces random associations between musical memories that create a new musical variation with the same style. The second or listening mode is needed to allow interaction with the current flow of on-going music structure by management of the knowledge stored in the memory.

In this chapter, we present two frameworks with first mode in common to both and that differ in how they handle the listening mode. In one framework, the listening functionality is accomplished by a human operator in charge of the virtual musical companion. In the second framework, this functionality is automatically handled by the machine through a collaborative, competitive and memory-based *Active Learning (AL)* process[4].

[4] It should be noted that the use of the term *Learning* in AL refers to the ability to adaptively select the best repertoire for improvisation (we refer to this as second mode), which is different from the learning aspect involved in construction of the musical dynamic memory that is central to the first mode of operation. The term learning refers here to learning of the criteria or the costs involved in selection of

In our concept, interaction is a cognitive process underlying an active learning of musical context done with reference to a finite memory of past and previous musical processes. With this distinction, interactivity between the system and musician goes beyond mere triggering and reoccurrence of past musical materials but is a combination of an ongoing acquisition of knowledge and memory of the musical process learned by the system and activated or deactivated through reinforcement by the musician in interaction with the system and vice versa. The underlying design and architectures presented here are inspired by cognitive research on musical expectations and their functionalities in constructing musical anticipations. We shall discuss this approach to operation of the style learning musical robot later in the paper.

In terms of interaction between the virtual performer and live musician, both modes are operational during improvisation, very much like in the case of interaction between human performers who alternate between listening and improvising, and whose improvisation might vary between recalling one's own materials or personal style versus imitating or fusion with other musician's style. In order to consider the musical and cognitive aspects of interaction with virtual performer, we shall term sometimes both modes indistinguishably as *stylistic reinjection*. In section 2 we will discuss the idea of stylistic reinjection and relate it to statistical modeling of musical sequences. Next, we will elaborate on one specific sequence representation that is currently used in several of our recent improvisation and composition systems, the Factor Oracle (FO) [8]. We will consider how new stylistically coherent sequences can be produced from FO analysis and representation and will show how the different generative parameters can influence the resulting musical outcomes. Factor Oracle constitutes the first mode mentioned above, i.e. learning and storage of a musical memory with stylistically important relations. After introducing FO, we will discuss the two frameworks presented separately. Section 5 discusses the issues of controlling and planning the productions using FO by a human operator. We then introduce the second framework in section 6, where an *Active Learning* process replaces the human operator for stylistic interactions. We conclude the chapter by reporting some of the different uses of the system and some musical performances.

2 Stylistic Re-injection

The musical hypothesis behind stylistic reinjection is that an improvising 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 instantaneous judgment 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

repertoire that would be appropriate for interaction, and it should be distinguished from the learning involved in forming the stylistic memory. The reason for using this term in AL is for its common use in the artificial intelligence literature.

present performance and of those by other performers are memorized, thus drifting back from present to the past. From the standpoint of long-term memory, these sound images can also act as inspiring sources of material that will eventually be recombined to form new improvised 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 improvisation interesting.

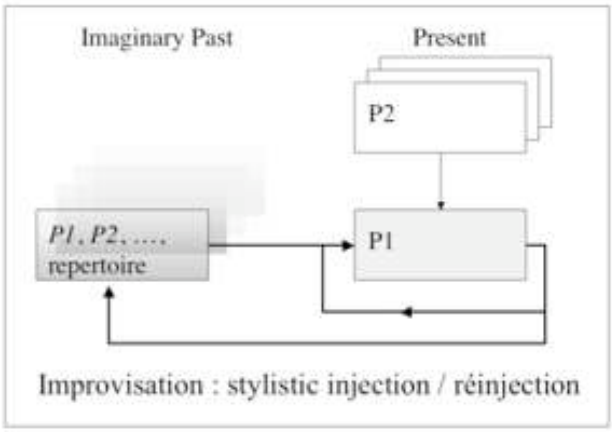


Fig. 1. Stylistic Reinjection.

The cognitive hypothesis behind stylistic reinjection has its roots in the psychology of music expectations [9]. Brain does not store sounds. Instead, it interprets, distills and represent sounds. It is suggested that brain uses a combination of several underlying presentations for musical attributes. A good mental representation would be one that captures or approximates some useful organizational property of a human's actual environment. But how does the brain know which representation to use? There is good evidence for existence of a system of rewards and punishments that evaluates the accuracy of our unconscious predictions about the world. Our mental representations are being perpetually tested by their ability to usefully predict ensuing events, suggesting that competing and concurrent representations may be the norm in mental functioning. In treating different representations and their expectations, each listener will have a distinctive listening history in which some representations have proved to be more successful than others. Accordingly, the conscious and unconscious thinking of an improvising performer constitutes the interactive level within which an external environment (through listening to others or himself) might influence the competitive and concurrent aspects of representations and learning of new representations. This interaction can be thus regarded as a reinforcement feedback

from an external environment onto the system that can activate an active learning process. The idea behind stylistic reinjection is to reify, using the computer as an external memory and leaning scheme, 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.

3 Statistical Music Modelling

Statistical modeling of musical sequences has been the subject of experimentation since the very beginning of musical informatics [10]. This chapter focuses on context models where 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 [11]; 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 given an alphabet Σ is $O(|\Sigma|^L)$, so only low order models are actually implemented.

To cope with the model order problem, in earlier works [12–15] 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 generation method. In what follows, 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 the next musical unit from the already generated sequence at each step. These prescription could be summarized as follows :

1. 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 playing musicians).
2. The generation of each musical unit must be bounded in time for compatibility with a real time scheduler.
3. 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.

4. 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.

For the frameworks presented here we chose a model named factor oracle (FO) [8] that conforms to points 1, 2 and 4 as the learning and storage for musical memory; described hereafter. Following FOs capacity of representation and storage, we extend the model to address the third point above in section 6.4.

4 Factor Oracle

The Factor Oracle concept comes from research on indexing string patterns. Such research has application in massive indexation of sequential content databases, pattern discovery in macromolecular chains, and other domains where data are organized sequentially. Generally stated, the problem is to efficiently turn a string of symbols S into a structure that makes it easy to check if a substring s (called a factor) belongs to S, and to discover repeated factors (patterns) in S. The relationship between patterns may be complex, because these are generally sub-patterns of other patterns; therefore the formal techniques and representations for extracting them, describing their relationships, and navigating in their structure are not obvious. However, these techniques are extremely useful in music research, as music at a certain level of description, is sequential and symbolic and the pattern level organization of redundancy and variation is central to its understanding.

Among all available representations (e.g. suffix trees, suffix automata, compression schemes), FO represents an excellent compromise, since

1. It computes incrementally and is linear in number of states and transitions
2. It is homogeneous, i.e. all transitions entering a given state are labeled by the same symbol, thus transitions do not have to be labeled, which saves a lot of space
3. It interconnects repeated factors into a convenient structure called SLT (suffix link tree)
4. Its construction algorithm is simple to implement, maintain and modify.

We will overview properties of FOs that are essential for our musical applications. Reader interested in more details about the algorithm should consult the original FO paper in [8].

Given a stream of symbols $s = \{s_1, s_2, \ldots s_n, \ldots\}$, the FO algorithm builds a linear automaton with (by convention left-to-right) ordered states $S_0, S_1, S_2 \ldots S_n$. Symbol s_i is mapped to state S_i and S_0 is an initial state (source). As said above, transitions in the FO are implicitly labeled by the symbol mapped to their target state. Forward (or factor) transitions connect all pairs of states (S_{i-1}, S_i), and some pairs (S_i, S_j) with $i < j - 1$. Starting from the source and following forward transitions one can build factors of s, or one can check if a string s' is a factor of s. We also consider some construction arrows named suffix links, used

internally by the FO algorithm for optimization purposes, which, however, have to be kept for musical applications. These backwards pointing arrows connect pairs of states (S_i, S_j) where $j < i$. A suffix link connects S_i to S_j iff j is the leftmost position where a longest repeated suffix of $s[1..i]$ is recognized. In that case, the recognized suffix of $s[1..i]$ – call it u – is itself present at the position $s[j - |u| + 1, j]$. Thus suffix links connect repeated patterns of s. Figure 2 shows the FO built from the string $s =$`abbbaab`. By following forward transitions, starting at the source, one can generate factors, such as bbb or aab. Repeated factors such as ab are connected through suffix links.

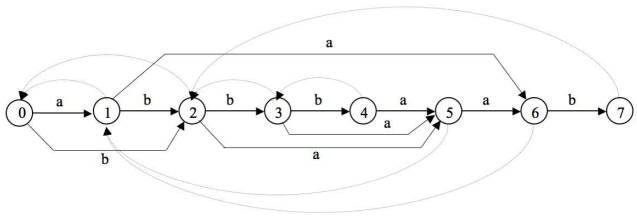


Fig. 2. The Factor Oracle for string `abbbaab`.

However, there is a problem with FO's. As can be checked on figure 2, the *false positive* factor `bbaa`, which is not present in s, can be generated as well. This is because the FO automaton does not exactly model the language of all factors of a string. They rather model a language that contains it. In other terms, if s' is a factor of s, it will be recognized as such. On the other hand, if s' is recognized, it is probably a factor of s.

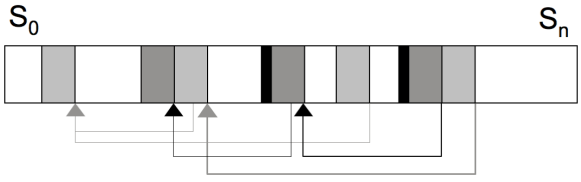


Fig. 3. Interconnection of repeated factors by suffix links.

Figure 3 shows how maximum length repeated factors are interconnected by suffix links. The thickness of the lines represents the length of the repeated factor. This length is computed at no additional cost by the oracle algorithm,

and we will see later that it provides a very important clue in the navigation. The colour of the lines (grey or black) separates two disjoint substructures in the set of suffix links, each of which forms a tree. The overall suffix links structure is a forest of disjoint trees, whose roots are the smallest and leftmost patterns appearing in the trees (see figure 4). A fundamental property of these Suffix Link Trees (SLT) is that the pattern at each node is a suffix of the patterns associated to its descendants (property 0). This way, the SLT capture all the redundancy organization inside the sequence.

Factor links also capture redundancy information, because of the following oracle property: let u a factor of s appearing at position i (position of its last symbol). There will be a factor link from i to a forward state j labelled by symbol a iff (property 1):

- u is the sub-word recognized by a minimal factor link path starting at the source 0 and ending in i.
- u is present at position $j-1$, forming the motif ua at position j.
- the motif ua at position j is the first occurrence of ua in the portion $s[i+1, |s|]$

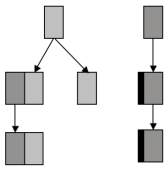


Fig. 4. Suffix links form a forest tree.

So a factor link connects two occurrences of a pattern, albeit with some restrictions.

5 Knowledge-based Interaction by a Human Operator

OMax is distributed across two computer music environments : OpenMusic and Max (see Figure 5). We will describe in more detail this architecture in last part of the paper. Here we outline the architecture of the improvisation system, without going into the implementation details. At this point we should mention that 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. 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 [16]. 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 the sound units into a continuous signal in real-time. Figure 5 shows a general diagram of this architecture.

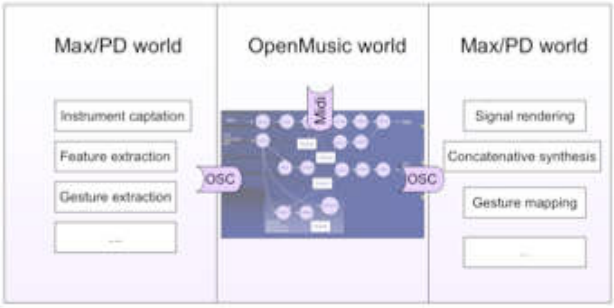


Fig. 5. OMax Architecture

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. In this paper, we limit our observations to the case of MIDI input and output whereas the underlying concepts can be easily expanded to audio units and structures as discussed in [17] and [18].

Inside the OpenMusic world exists a community of concurrent agents that can be freely instantiated and arranged into different topologies. These agents belong to six main classes : (1) Listener, (2) Slicer, (3) Learner, (4) Improviser, (5) Unslicer and (6) Player.

Figure 6 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 generates a stream that is prepared to be fed into the rendering engine in Max.

The statistical modeling techniques we use supposes 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 [14] a method for processing polyphonic

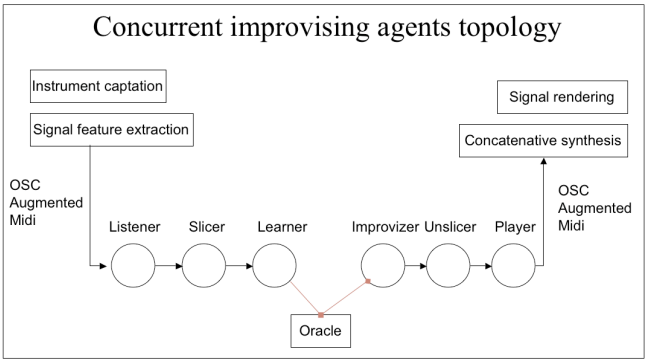


Fig. 6. A simple agent topology.

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" are dedicated to transforming the raw augmented MIDI stream into the slice representation used by the model, then back into a polyphonic stream (see Figure 7).

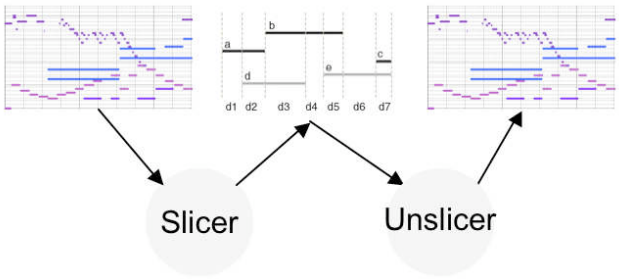


Fig. 7. Slicing Polyphony

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

5.1 A Simple OMax Topology

For example, in Figure 8, 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 result in generating 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.

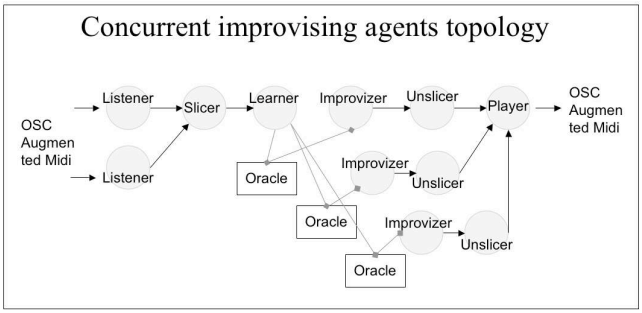


Fig. 8. Another Agent Topology

5.2 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 9, 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 recreate 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 manner.

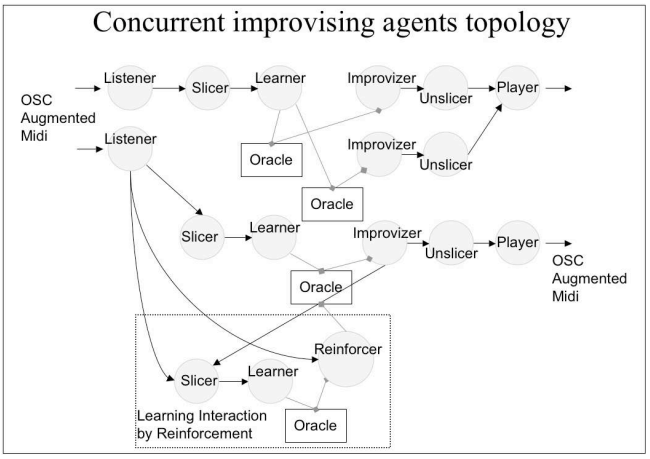


Fig. 9. An agent topology with meta-oracle

6 Knowledge-based Automatic Interaction

In this section, we present another architecture where the stylistic interaction is automatically controlled by the machine itself and without a human operator. In previous works, interaction between an improviser machine and human performer has been limited to generation of relevant musical sequences by the computer that can be regarded as a continuation of the previous context generated by the human performer. As mentioned earlier, the continuation is assured by a stylistic reinjection of past material (through recombination) that has the most predictive value. In this work, we extend this view, philosophically speaking, by putting forth the concept of *musical anticipation* in the social context of musical improvisation. Within this view, interaction is viewed as the concurrent use of knowledge and memory where the *anticipatory values* of actions replace the predictive values. Knowledge is thus gained by constant interaction between the improvising machine and the performer (or a score in case of pure generation) and also by a constant self-reflection of the system by listening to its own generation. It is evident that such an automatic interactive procedure requires a second learning module than the structure learning of FO described earlier. The learning procedure described hereafter, updates anticipatory values to relevant memory states stored in the system. These values are thus in constant change during an improvisation session whether the system is generating alone or improvising with a human performer. They enable access to the relevant parts in

the memory given the current real-time context. These values are anticipatory rather than predictive since they evaluate the relevance of each state for generating the longest contextually meaningful sequence in the future. Learning is thus memory-based and its access to past structure is done through activation of previous states by evaluating them versus the current real-time context from the environment, thus the term Active Learning (AL).

The main inspiration behind this framework comes from cognitive foundations of music expectations as discussed in section 2. The complexity of an improvisation process is beyond one's imagination, which makes it impossible to model in terms of computer structures and logics within the structure. In our conception, the complexity of structure, as Simon puts it [19], is not due to the complexity of the underlying *system* itself but due to the complexity of its environment. Hence, an architecture that aims at modeling the complexity of an improvisation system must be *adaptive* and demonstrate *intelligence* for learning and adopting reactivity between the agents and the environments when desired. The framework presented here is a first attempt in modeling such interactions.

6.1 Active Learning

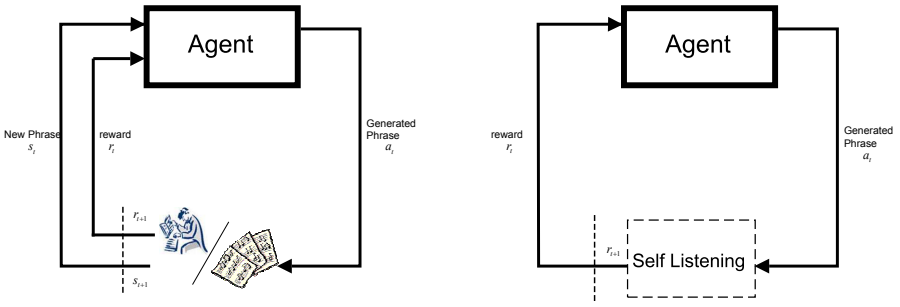
There are two main reasons for our consideration of Active Learning (AL) algorithms (to be defined shortly). The first, being an *enactive* view of music cognition, emphasizes the role of *sensory-motor* engagement in musical experience. The enactive view of cognition and the link between perception and action [20] is dominated by concerns with visual experience. If for the sake of simplicity and coherence of this presentation we set aside the visual and gestural aspects of a music performance, the sense of an auditory sensory-motor knowledge becomes less obvious than visual experience. The point here is that perceptual experience is *active* and thoughtful. As Wessel puts it correctly in [21], one can imagine an auditory version of the classic perception action link experiments by Held and Hein [22] where it is shown that that a kitten with a passive exploration experiment of an environment has considerable perceptual impairment compared to the one who was actively transacting with the environment. In the context of an stylistic interaction with a performer, the interaction is nothing but an active exploration and exploitation of the constantly changing environment where the sensory-motor knowledge takes the form of *conceptual understanding* [20].

The second and more technical motivation behind this choice, is the fact that in an improvisation setting and for an agnostic system such as ours with no human intervention, the *learner* should have the ability to influence and select its own training data; hence the general definition of the *active learning* problem. An early landmark research on this topic is the selective sampling scheme of Cohn, Atlas and Ladner [23] which became the main inspiration for many subsequent works in the field.

The Active Learning algorithm presented here has close ties to the Reinforcement Learning (RL) paradigm [24] and is a continuation of a previously presented work using RL algorithms in [25]. In a regular RL framework, learning is performed by simulating episodes of state-action pairs and updating the

policies in order to obtain a maximum reward towards a defined goal. Therefore, RL systems undergo goal-oriented interaction. In our design, defining a goal would be either impossible or would limit the utility of the system to certain styles. As will be explained shortly, what is traditionally regarded as rewards in an RL framework is termed *guides* in our proposed framework and helps to direct learning updates to relevant states in the memory pertaining to the current environmental context. Note that while the simulation episodes during learning of RL algorithms is a way to solve the burden of unsupervised learning, it does not help the agents to explore the environment actively and would usually require sufficiently large amount of time so that all (relevant) states are visited and updated accordingly. On the contrary, in an Active Learning framework exploration is enhanced by guiding the learning agents to the relevant states in the memory given a context where changes are most probable. A global description of the AL algorithm is shown hereafter.

Figure 10 demonstrate block diagrams for two modes of the proposed Active Learning algorithm. In the first, referred to as Interaction Mode, the system is interacting with a human performer (or an incremental read of a music score). In this mode, environmental signals consist of musical sequences generated by the human performer (or the last read musical phrase from a score). In the second mode, referred to as self-listening, the system is in the generation phase and is interacting with itself, or in other words listening to itself so that the environmental signals are a feedback of the last generated sequence onto the agent itself.



(a) Interaction Mode (b) Self Listening Mode

Fig. 10. Machine Improvisation modes diagram

The interaction mode occurs when external information is being passed to the system from the environment (human improviser). This way the reward would be the manner in which this new information reinforces (positively or negatively) the current stored models in the agent. The self listening mode occurs when the system is improvising new sequences. In this mode, the AL framework would be in a model-free learning state, meaning that the internal model of the

environment stays intact but the values of actions are influenced by the latest musical sequence that has been generated by the system itself, thus, the idea of self listening.

Once the *anticipatory values* corresponding to each action in the memory is learned, generation amounts to choosing randomly between the *best* actions on a state weighted by their anticipatory values.

The Active Learning framework presented here consists of three main modules each with independent but coherent tasks and is inspired by the *Dyna* architecture [24]. The modules are as follows:

1. *Model Learning:* Constitutes a state-space representation of the past knowledge observed by the system and enables access to previous state-action paths whenever desired by active learning or generation modules.
2. *Guidage:* Responsible for transactions between the environment and the improvisation agent. In other words, *Guidage* guides the agent to *relevant* state-action pairs stored in previously learned models based on the current environmental context.
3. *Anticipatory Learner:* At each interaction with the environment (through *Guidage*), anticipatory values corresponding to each state-action pair (in each model) is learned through a competitive, concurrent and memory-based learning.

Algorithm 1 shows the main interactive cycle of the Active Learning algorithm. This architecture uses learning cycles with multiple agents constituting competitive and concurrent models. These cycles happen at each interaction between the system and the environment and uses an active learning module instead of the traditional reinforcement learning. Upon the arrival of environmental or interactive sequences, the system prepares for learning by (1) calculating *immediate rewards* for stored states in system's memory and (2) selecting relevant states for the anticipatory learning procedure. This preparatory procedure, responsible for interactivity of the system, is referred to as *Guidage* and will be discussed in section 6.3. The memory models are based on the FO representation, as described in section 4. In order to use these models in the context of reinforcement learning, we will view them as a particular case of so called Markov Decision Processes, described in section 6.2. These models are then appropriately learned and updated in case of being in Interaction Mode. After this step, the system is ready to update all the anticipatory profiles of stored states in parallel agents through competitive, concurrent and memory-based learning described in sections 6.4.

6.2 Model Learning

The agent in both modes consists of a model-based AL 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 environmental elements. Another issue comes from the fact that music information has a

Algorithm 1 `Active Learning` Interactive Cycle

Require: At each time t: previously learned models (FO_{t-1}), the new environmental sequence $A^t = \{A_1, A_2, \cdots, A_N\}$

1: Obtain active states and guides using `Active Selection` and A^t and FO_{t-1}s.
2: **if** we are in *Interaction Mode*, **then**
3: Update Models through learning (FOs)
4: **end if**
5: Perform Competitive, Concurrent and Memory-based Active Learning

natural componential and sequential structure. While sequential models have been extensively studied in the literature, componential or multiple-attribute models still remain a challenge due to complexity and explosion in the number of free parameters of the system. Therefore, a significant challenge faced with music signals arises from the need to simultaneously represent and process many attributes of music information. The ability (or inability) of a system to handle this level of musical complexity can be revealed by studying its ways of musical representations or memory models both for storage and learning. The main feature of the agent is that it tackles this multi-componential aspect of musical representations (or viewpoints) as separate agents in the architecture. The AL algorithm used would then be a collaborative and competitive learning between viewpoints.

Any representation chosen for this task, should be compact, informative, incrementally learnable and should provide an easy access through memory for later use. Factor Oracles, discussed earlier, have been chosen to represent and store musical structures pertaining to some music features. Here we present the use of FOs in a Markov Decision Process (MDP) framework suitable for learning and generation. An MDP in general is defined by a set of states-action pairs $S \times A$, a reward function $R : S \times A \rightarrow \mathbb{R}$ and a state transition function $T : S \times A \rightarrow S$. To conform the representational scheme presented before to this framework, we define MDP state-action pairs as FO states and emitted symbol from that state. The transition function would then be the deterministic FO transition functions as defined before.

In order to construct and learn models incrementally, music MIDI signals are first parsed as shown in figure 11 and multi-dimensional features for each event is extracted as shown in table 1 for this sample. For this experiment, we use pitch, harmonic interval and beat duration and their first derivatives (thus a total of 6) multiple representations.

Once this data is collected, learning Factor Oracle is straightforward as discussed in section 4. In our framework, each feature vector (each row in table 1) is represented in different FOs as is demonstrated in figure 12 for four rows corresponding to data in table 1.

These representations constitute the model-learning for concurrent and collaborative agents at work in the Active Learning framework and provide a compact model of the musical memory that will be used during learning and generation. As stated earlier, FO structures can be updated incrementally. Addition

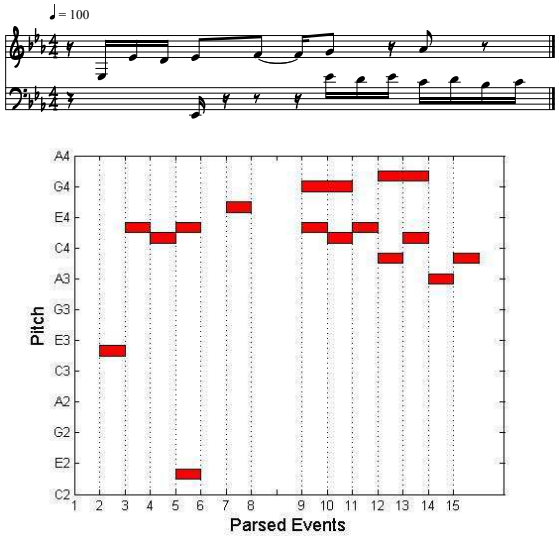


Fig. 11. Parsed pianoroll presentation for the first measure of J.S.Bach's *two-part Invention No.5* (Book II) with quantization of $\frac{1}{16}$ beats

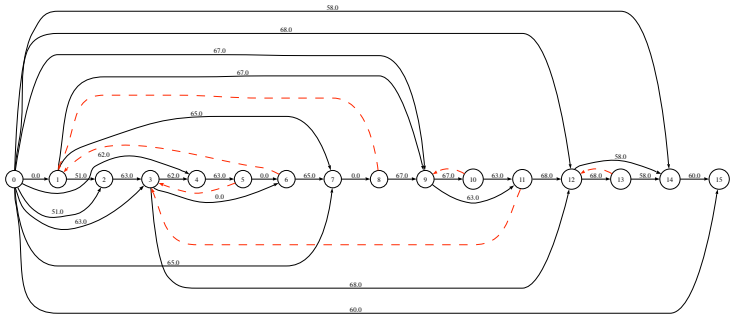
Event Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Pitch (MIDI)	0	51	63	62	63	0	65	0	67	67	63	68	68	58	60
Harmonic Int.	0	0	0	0	24	0	0	0	4	5	0	8	6	0	0
Duration	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1
Pitch Diff..	0	0	12	-1	1	0	0	0	0	0	1	-3	0	-4	2
Harm. Diff.	0	0	0	0	0	0	0	0	0	1	0	0	-2	0	0
Dur. Ratio	1	1	1	1	1	1	1	2	0.5	1	1	1	1	1	1

Table 1. Time series data on the score of figure 11 showing features used in this experiment

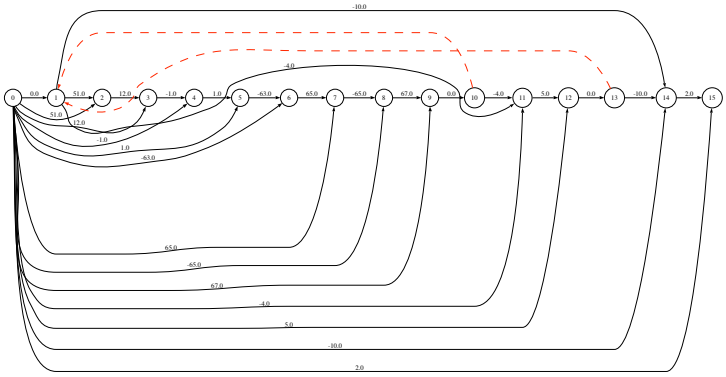
and learning of new FO structure happens only during the *Interaction Mode* (figure 10) and the models stay intact during *self-listening* mode of the system. For a discussion on the compactness and complexity of FO representation compared to other available algorithms we refer the reader to [25].

6.3 Guidage

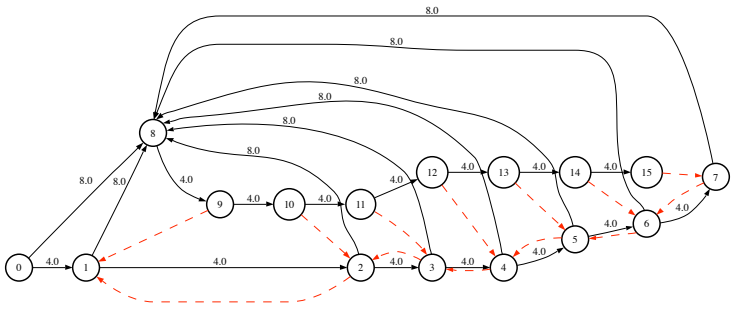
At each interaction of the system with the environment (through reception of a musical sequence or self-listening of previous generated sequence), the system should adapt itself to the current *context* and learn the new behavior. In our Active Learning framework, this amounts to selection of relevant states in each memory models discussed previously and assigning appropriate *immediate reward* signals to each selected state. This process is undertaken upon each in-



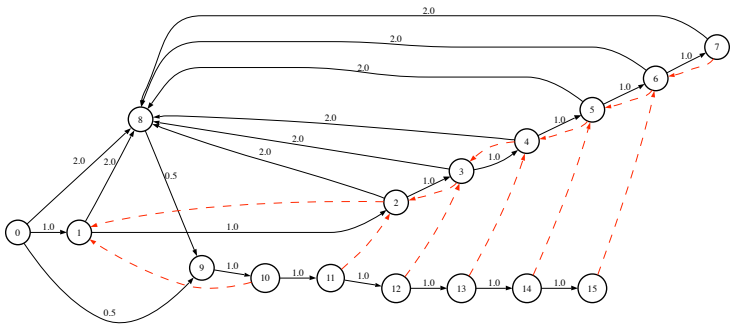
(a) Pitch FO



(b) Pitch Contour FO



(c) Duration FO



(d) Duration Ratio FO

Fig. 12. Learned Factor Oracles over pitch and duration sequences in table of table 1. Each node represents a state, each solid line a *transition* and dashed line a *suffix link*.

teraction with the outside environment and guides the anticipatory learner to appropriate state-action pairs in learned models, hence the name *Guidage*.

In a 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. The reward signals calculated during the active selection procedure signify not only the immediate reward obtained by performing the appropriate state-action pair, but also *guide* the learning in the next procedures to the appropriate places in the memory where anticipatory values are most likely to change and need adaptation. This way, the *Guidage* procedure becomes a search algorithm adapted to FO structures that assign immediate rewards

The search algorithm proposed here is based on Dynamic Programming, an algorithm paradigm in which a problem is solved by identifying a collection of subproblems and tackling them one by one, smallest first. Dynamic Programming uses the "answers" to small problems to help figure out larger ones, until the whole of them is solved. In the context of our problem, the "small" problem set amounts to finding musical phrases (or combinations thereof) in the previously learned models, that are *similar* to the ongoing context from the environment *and* can be considered a *continuation* of the previously obtained chained based on the Factor Oracle structure in the memory. We call this step of the procedure the *forward pass*. The "larger" problem, then, becomes finding the *best path* among all recognized that best meets the search criteria when all the small-set problems are solved. This step is referred to as *backward* procedure.

To describe the algorithm in a formal manner, we use the following notations: Input stream from the environment is parsed into individual features represented as $A^k = \{A_1^k, A_2^k, \ldots, A_N^k\}$ where each each A_i^k is the k^{th} feature (or viewpoint) that corresponds to the k^{th} factor oracle with time index i. Similarly, the search target FO is represented by its stored states $s_i : 0 < i \leq M$. Using these notations, the forward pass of *Guidage* returns a structure I, which refers to found indexes on a Factor Oracle structure that can be regarded as an analogy to the score matrix in a dynamic programming paradigm. Within I, subsets $I_i = \{I_i^1, \ldots, I_i^k\}$ contain several paths associated with target FO states and refer to the possible *reconstruction indexes* on the target FO that can reassemble the whole or factors of the environmental sequence.

Having I, finding the *best paths* amounts to a backtracking procedure over subsets I_i, resulting into a *reconstruction matrix* $R_M : N \times \ell$. Here, N refers to the length of the input stream and ℓ is the *search depth* or the longest sequence that has achieved during search operation. Similar to I, each row k in the reconstruction matrix R corresponds to a sub-phrase in the environmental sequence and the contents of that row are index references to the Factor Oracle state-actions stored in the memory. These indexes correspond to state-action pairs in the model that can *reassemble* the "longest possible factor" within the environmental sequence up to frame k. Obviously, in the case of self-similarity for an environmental sequence of length N, the algorithm results to an $N \times N$

symmetric reconstruction matrix R with each M^{th} row ($M \leq N$) in the form $\{M, M-1, \ldots, 2, 1, 0, \ldots, 0\}$.

Once the reconstruction matrix R_M is obtained, *immediate rewards* to each state indicated in I are assigned as depth of the reconstruction relative to the total input length. For example, the reward for $\{s_{1^*} s_{2^*} \ldots s_{N^*}\}$ would be equal to 1 if the state/transition path $s_{1^*} \ldots s_{N^*}$ regenerate literally the sequence A^t.

Rewards or guides are calculated the same way for both modes of the system described before with an important difference. We argue that the rewards for the *interaction mode* (Figure 10(a)) correspond to a *psychological attention* towards appropriate parts of the memory and guides for the *self-listening mode* (Figure 10(b)) correspond to a *preventive* anticipation scheme. This means that while interacting with a live musician or sequential score, the system needs to be attentive to input sequences and during self-listening it needs to be preventive so that it would not generate the same path over and over. Moreover, these schemes provide the conscious reinforcement required to encourage or discourage useful and useless thinking as mentioned in section 2. This is achieved by treating environmental rewards with positive and negative signs appropriately.

In summary, after the arrival of an environmental sequence (either through self-listening or interaction with a musician/score), *Guidage* evaluates the relevancy of past memory states to the new sequence resulting into the reconstruction matrix R_M and also returns immediate reward values or guides for each of those states determining the degree of relevance. These indexed states and their guides will be used during anticipatory learning to adapt the behavior of the system to the current musical context.

6.4 Anticipatory Learning

The *Guidage* procedure described earlier, provides *immediate rewards* for undertaking different actions (emitting musical symbols) for each state in the stored memory model. While these values can provide instant continuations for a generative process given a context, they simply fail to maintain a coherent context-based structure in longer temporal structures. In order to maintain this long-term coherency during generation either a human-operator or an external learning agent is needed to optimize the temporal context of the generated sequence. The idea behind *Anticipatory learning* is to further enhance the knowledge of the system by blending the ongoing immediate guides with their values in an infinite future horizon.

Given FO representations as Markov Decision Processes, Active Learning procedure aims at finding policies as a mapping probability $Q(s, a)$. This way the policy can be represented as a matrix Q which stores values for each possible state-action pair in a given FO. The Q-learning algorithm proposed here is very similar to its Reinforcement Learning analogy except that we run updates on preselected states from *Active Selection* instead of running simulation episodes.

As discussed in section 2, different mental representations of music work in a collaborative and competitive manner based on their predictive power to make decisions. This can be seen as kind of a *model selection* where learning uses all the

agents' policies available and chooses the best one for each episode. This winning policy would then become the *behavior policy* with its policy followed during that episode and other agents being influenced by the actions and environmental reactions from and to that agent.

At the beginning of each learning cycle, the system selects one agent using a Boltzmann equation over the previously learned Q values on the starting state s_0. This way, a behavior policy π^{beh} is selected *competitively* at the beginning of each cycle based on the predictive value of the initial state s_0 among all policies π^i. To verbalize the procedure, the idea here is to efficiently detect the prominent behavior of the latest incoming signal from the environment. For example, if the latest phrase from the environment constitutes a highly rhythmical profile, naturally the agent corresponding to rhythmic feature should be selected as the behavior policy.

During each learning cycle, the agents would be following the behavior policy. For update of π^{beh} itself, we use a simple Q-learning algorithm [24] but in order to learn other policies π^i, we should find a way to compensate the mismatch between the target policy π^i and the behavior policy π^{beh}. Uchibe and Doya [26] use an *importance sampling* method for this compensation and demonstrate the implementation over several RL algorithms. Adopting their approach, during each update of π^i when following π^{Beh} we use a compensation factor based on importance sampling. During Q-learning, the Q value associated with a state-action pair (s_i, a_i) (in viewpoint i) is updated by the following factors: (1) An *infinite horizon reward* factor $R(s)$, different from the immediate reward $r(.,.)$ discussed in section 6.3, where the idea comes from our interest in the impact of future predictions on the current state of the system. This means that the reward for a state-action pair would correspond to future predicted states. and (2) a regular RL *Time Difference* update influenced by the mentioned importance sampling between-viewpoints' compensation factor.

This scheme defines the *collaborative* aspect of interactive learning. For example, during a learning episode, pitch attribute can become the behavior policy Q^{beh} and during that whole episode the system follows the pitch policy for simulations and other attributes' policies $Q^i(.,.)$ will be influenced by the behavior of the pitch policy.

In the Q-learning algorithm above, state-action pairs are updated during each cycle and on relevant indexes found by *Guidage*. This procedure updates one state-action pair at a time. In an ideal music learning system, each immediate change should evoke previous related states already stored in the memory. In general, we want to go back in the memory from any state whose value has changed. When performing update, the value of some states may have changed a lot while others rest intact, suggesting that the predecessor pairs of those who have changed a lot are more likely to change. So it is natural to prioritize the backups according to measures of their urgency and perform them in order of priority. This is the idea behind *prioritized sweeping* [27] embedded in our learning. In this procedure, each update is examined through a threshold if the

amount of value change is judged significant, all state-actions leading to that state will be added to a list of further updates in a recursive manner.

7 Current Uses and Results

7.1 OMAX with sound: OFON extensions

The Ofon system was developed in order to use the Omax system with acoustic instruments instead of Midi. Current version only works with monophonic instruments, where a pitch tracker is used to extract the pitch and intensity values represented as Midi, while the real sound of the instrument is recorded into an audio buffer. The system uses the `yin` pitch algorithm [28] as implemented in Max by Norbert Schnell, along with some post processing of our own.

Using time stamps at the offset of sound events, a sequence of symbols corresponding to pitch events are learned in the FO in a reference to a sound buffer. When OM performs the reconstruction of an improvisation using the oracle, the time stamps are sent back to Max along with the Midi stream. An Ofon module, the audio montage, is then able to assemble the sound pieces extracted from the sound buffer thanks to the time stamps. So, the music is learned from an audio signal, and the virtual improvisations result in an audio signal reconstructed from that original audio material. It is worth noting that, since we can get several computer-generated improvisations going on at the same time, it becomes possible to create multiple clones of the live player and have him/her play with them. This results, of course, in an exciting interaction situation. For the moment, the system has been extensively used with saxophone players (Philippe Leclerc in France and Tracy McMullen and David Borgo in San Diego), and experimented with student actors of the Théâtre national de Strasbourg in a language and sound poetry environment under the direction of the stage director Jean-François Peyret. Figure 13 shows one such concert.

Fig. 13. An Ofon concert at Ircam (SMC'05, Workshop on Improvisation with the Computer) : left, Philippe Leclerc, saxophone. In the center, "la Timée. Right, Olivier Warusfel and Georges Boch.

7.2 OFon Video

The principle of *Ofon-video* is the same as in Ofon. The performer is filmed while the learning is performed on the music. Then the filmed extracts are re-edited in real-time, according to the Omax virtual improvisation sequences labeled by time stamps. This creates a recombined improvisations of the music combined with their respective recombination of video sequences. The system was developed by three students of University of Strasbourg II (Anne-Sophie Joubert, Vincent Robischung and Émilie Rossez) under the direction of Georges Bloch. Ofon-video has been programmed with jitter, an extension of the Max programming language aiming primarily at video control.

One of the main interests of Ofonvideo is to allow improbable encounter between musicians. For example, the saxophonist Philippe Leclerc can meet Thelonius Monk, improvise with him, and each musician influences the improvisation of the other. The filming ends up in a bottomless mirror, in which Leclerc watches Leclerc who watches Monk who watches Leclerc... see figure 14.

Fig. 14. A picture of an Ofonvideo session, with Philippe Leclerc, saxophone, watching himself watching Monk. This very picture may appear seconds later on the screen.

More examples (audio, midi and video) can be found at

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

7.3 Active Learning Generation

There are many ways to generate or improvise once the policies for each attribute are available. We represent one simple solution using the proposed architecture. At this stage, the system would be in the *self listening mode* (Figure 10(b)). The agent would generate *phrases* of fixed length following a behavior policy

(learned from the previous interaction). When following the behavior attribute, the system needs to *map* the behavior state-action pairs to other agents in order to produce a complete music event. For this, we first check and see whether there are any common transitions between original attributes and, if not, we would follow the policy for their derivative behavior. Once a phrase is generated, its (negative) reinforcement signal is calculated and policies are updated as in section 6.1 but without updating the current models (FOs).

Audio results of automatic improvisation sessions on different styles can be heard on the internet[5]. As a sample result for this paper, we include analysis of results for style imitation on a polyphonic piece, *two-part Invention* No.3 by J.S. Bach. For this example, the learning phase was run in *interaction mode* with a sliding window of 50 events with no overlaps over the original MIDI score. After the learning phase, the system entered *self listening* mode where it generates sequences of 20 events and reinforces itself until termination. The generated score is shown in Figure 15 for 240 sequential events where the original score has 348. For this generation, the *pitch* behavior has *won* all generation episodes and direct mappings of *duration* and *harmonic* agents have been achieved 76% and 83% in total respectively leaving the rest for their derivative agents.

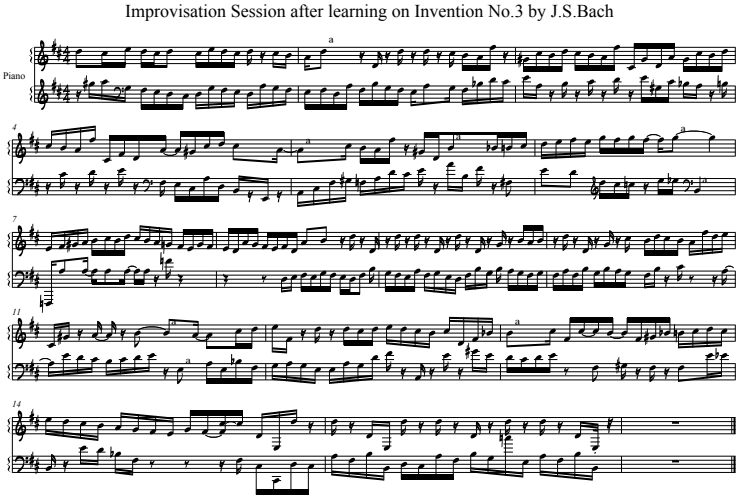


Fig. 15. Style imitation sample result

While both voices follow a polyphonic structure, there are some formal musicological structures that can be observed in the generated score. Globally, there are *phrase* boundaries in measures 4 and 11 which clearly segment the score into three formal sections. Measures 1 to 4 demonstrate some sort of exposition of musical materials which are expanded in measures 7 to the end with a transition

[5] http://cosmal.ucsd.edu/∼arshia/

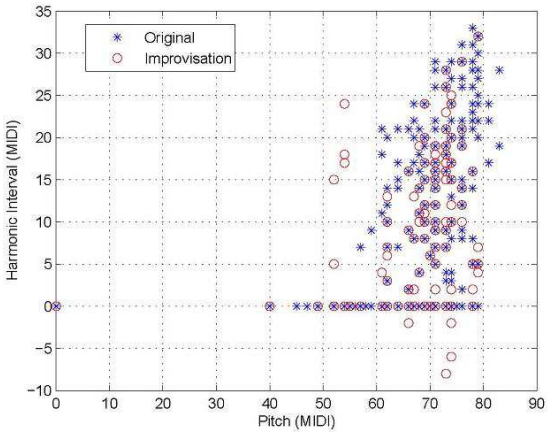


Fig. 16. Improvisation Space vs. Original Space

phase in measure 5 and 6 ending at a week cadence on `G` (a fourth in the given key). There are several thematic elements which are reused and expanded. For example, the repeated `D` notes appearing in measures 2 appear several times in the score notably in measure 7 as low A with a shift in register and harmony and measure 9 and 15. More importantly, these elements or their variants can be found in the original score of Bach. Figure 16 shows the pitch-harmony space of both the original MIDI and the generated score. As is seen, due to the collaborative and competitive multi-agent architecture of the system, there are new combinations of attributes which do not exist in the trained score.

References

1. Chabot, X., Dannenberg, R., Bloch, G.: A workstation in live performance: Composed improvisation. In: International Computer Music Conference (ICMC). (Octobre 1986) 537–540
2. Zicarelli, D.: M and jam factory. Computer Music Journal **11**(4) (Winter 1987) 13–29
3. Lewis, G.: Too many notes: Computers, complexity and culture in voyager. Leonardo Music Journal **10** (2000) 33–39
4. Wessel, D., Wright, M.: Problems and prospects for prospects for intimate musical control of computers. In: New Interfaces for Musical Expressions (NIME). (2001)
5. Biles, J.A.: Genjam in perspective: A tentative taxonomy for genetic algorithm music and art systems. Leonardo **36**(1) (2003) 43–45
6. Pachet, F.: Interactions réflexives. In Orlarey, Y., ed.: Actes des recontres musicales pluridisciplinaires, Lyon, Grame (2006)
7. Assayag, G., Bloch, G., Chemillier, M.: Improvisation et réinjection stylistique. In Orlarey, Y., ed.: Actes des recontres musicales pluridisciplinaires, Lyon, Grame (2006)

8. Allauzen, C., Crochemore, M., Raffinot, M.: Factor oracle: A new structure for pattern matching. In: Proc. of Conference on Current Trends in Theory and Practice of Informatics, London, Springer-Verlag (1999) 295–310
9. Huron, D.: Sweet Anticipation: Music and the Psychology of Expectation. MIT Press (2006)
10. Conklin, D.: Music generation from statistical models. In: Proceedings of Symposium on AI and Creativity in the Arts and Sciences. (2003) 30–35
11. Ron, D., Singer, Y., Tishby, N.: The power of amnesia: Learning probabilistic automata with variable memory length. Machine Learning **25**(2-3) (1996) 117–149
12. Dubnov, S., Assayag, G., El-Yaniv, R.: Universal classification applied to musical sequences. In: Proc. of ICMC, Michigan (1998) 322–340
13. Assayag, G., Dubnov, S., Delerue, O.: Guessing the composer's mind : Applying universal prediction to musical style. In: ICMC: International Computer Music Conference, Beijing, China (Octobre 1999)
14. Dubnov, S., Assayag, G., Lartillot, O., Bejerano, G.: Using machine-learning methods for musical style modeling. IEEE Computer Society **36**(10) (2003) 73–80
15. Assayag, G., Dubnov, S.: Using factor oracles for machine improvisation. Soft Computing **8-9** (2004) 604–610
16. Wright, M.: Open sound control: an enabling technology for musical networking. Organised Sound **10**(3) (2005) 193–200
17. Dubnov, S., Assayag, G., Cont, A.: Audio oracle: A new algorithm for fast learning of audio structures. In: Proceedings of International Computer Music Conference (ICMC), Copenhagen (September 2007)
18. Cont, A., Dubnov, S., Assayag, G.: Guidage: A fast audio query guided assemblage. In: Proceedings of International Computer Music Conference (ICMC), Copenhagen (September 2007)
19. Simon, H.A.: The Science of the Artificial. MIT Press (1969)
20. Noë, A.: Action in Perception. MIT Press, MA (2004)
21. Wessel, D.: An enactive approach to computer music performance. In Orlarey, Y., ed.: Actes des recontres musicales pluridisciplinaires, Lyon, Grame (2006)
22. Held, R., Hein, A.: Movement-produced stimulation in the development of visually guided behavior. In: Journal of Comp. Physiol. Psych. Volume 56. (Octobre 1963) 872–6
23. Cohn, D.A., Atlas, L., Ladner, R.E.: Improving generalization with active learning. Machine Learning **15**(2) (1994) 201–221
24. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
25. Cont, A., Dubnov, S., Assayag, G.: Anticipatory model of musical style imitation using collaborative and competitive reinforcement learning. In M.V., B., O., S., G., P., G., B., eds.: Anticipatory Behavior in Adaptive Learning Systems. Volume 4520 of LNAI. Springer Verlag, Berlin (2007) 285–306
26. Uchibe, E., Doya, K.: Competitive-cooperative-concurrent reinforcement learning with importance sampling. In: Proc. of International Conference on Simulation of Adaptive Behavior: From Animals and Animats. (2004) 287–296
27. Moore, A., Atkeson, C.: Prioritized sweeping: Reinforcement learning with less data and less real time. Machine Learning **13** (1993) 103–130
28. de Cheveigné, A., Kawahara, H.: YIN, a fundamental frequency estimator for speech and music. J. Acoust. Soc. Am. **111** (2002) 1917–1930