Using Factor Oracles for Machine Improvisation

Gérard Assayag, Ircam, assayag@ircam.fr
Shlomo Dubnov, UCSD, sdubnov@ucsd.edu

***Abstract** : we describe variable markov models we have used for statistical learning of musical sequences, then we present the factor oracle, a data structure proposed by Crochemore & al for string matching. We show the relation between this structure and the previous models and indicate how it can be adapted for learning musical sequences and generating improvisations in a real-time context.*

Modeling musical sequences

Statistical modeling of musical sequences has been experimented since the very beginnings of musical informatics (see [Con03] for a review and criticism of most existing models, and [Zic87] for one of the first available real-time interactive system). The idea behind *context models*, which we are mostly interested in here, 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 sequence. For example, if w is musical sequence, σ a symbol belonging to the musical alphabet Σ, $P(\sigma|w)$ is the probability that σ will follow w, i.e. the probability of $w\sigma$ given w. This distribution P 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. [Ron&al96] explain that this idea dates back to Shannon : complex sequences do not have obvious underlying source, however, they exhibit a property called *short memory property* by the authors; there exists a certain memory lengh L such that the conditional probability distribution on the next symbol σ does not change significantly if we condition it on suffixes of w 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 experimented.

To cope with the model order problem, in earlier works [Dub98,Dub02,Dub03,Ass99] 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).

The IP method is derived from Information Theory. J. Ziv and A. Lempel [Ziv78] first suggested the core of this method called Incremental Parsing in the context of lossless compression research. IP builds a dictionary of distinct motifs by making a single left to right traversal of a sequence, sequentially adding to a dictionary every new phrase that differs by a single last character from the longest match that already exists in the dictionary. Using a tree representation for the dictionary, every node is associated with a string, whose characters appear as labels on the arcs that lead from the root to that node. Each time the parsing algorithm reaches a longest-match node it means that the node's string has already occurred in

the sequence. Then IP grows a child node, with an arc labeled by the next character in the sequence. The new node denotes a new phrase that differs by one last character from its parent. [Fed03] has proved that an universal predictor outperforms asymptotically (when the sequence length grows to infinity) any Markov predictor of a finite order L. Furthermore, at a given stage, the dictionary representation stores nodes that are associated with strings of length from 1 to L, where L is the depth of the IP Tree. These nodes have the same meaning as Markov states; only their number is dramatically smaller than the number of states that would be needed by a regular L-order Markov model.

[Ron&al96] suggested a different VMM structure called Prediction Suffix Tree (PST), named after the data structure used to represent the learned statistical model. PST represents a dictionary of distinct motifs, much like the one generated by the IP algorithm. However, in contrast to the lossless coding scheme underlying the IP parsing, the PST algorithm builds a restricted dictionary of only those motifs that both appear a significant number of times throughout the complete source sequence, and are meaningful for predicting the immediate future. The framework underlying the approach is that of efficient lossy compression.

One may note that both IP and PST build tree structures in the learning stage, where finding the best suffix consists of walking the tree from the root to the node bearing that suffix. The main difference between the methods is that IP operates in the context of lossless compression, cleverly and efficiently sampling the string statistics in a manner that allows a compressed representation and exact reconstruction of the original string. PST, which was originally designed for classification purposes, has the advantage of better gathering of statistical information from shorter strings, with a tradeoff of deliberately throwing away some of the original sub-strings during the analysis process to maintain a compact representation (thus being a "lossy" compression method), as well as allowing for a small probability production for all possible continuations for any given suffix.

We have carried extensive experiments on using IP for music classification and music generation. We have also implemented a version of PST's adapted to music and compared the results with IP. These experiments are described in [Dub03]. From these experiences we can draw a series of prescriptions for a 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 the model and generates a musical stream by predicting at each step the next musical unit from the already generated sequence. Depending on the specific application, learning and generating can be off-line or on-line, consecutive or threaded, real-time or non real-time. Of course the real-time application is the more demanding, so we will specify the following prescriptions for a real time improvisation system:

1. Learning must be incremental and fast in order to be compatible with real-time interaction, and switch instantly to generation (real-time alternation of learning and generating can be seen as « machine improvisation » where the machine « reacts » to other musician playing).
2. The generation of each musical unit must 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 (e.g. IP, PST, others) and switch between them at generation time.
4. In order to cope with the parametric complexity of music (multi-dimensionality and multi-scale structures) multi-attribute models must be searched for.

As for point 1., IP is fine, but PST does not conform [Dub03].

In order to comment on point 2, some precision on the generation process must be given. Whatever model is chosen, a generation step is as follows :

> let *w* be the sequence generated so far, let *w = vu* where u is the *best suffix* of *w*, that is the longest string that can be find associated to a node in the model. *u=e* and *u=w* are possible situations. There is a conditional probability distribution P associated to the node, which, for every symbol σ in Σ gives the probability $P(\sigma|u)$ that σ follows *u*. Let σ' be a stochastic choice drawn from Σ with respect to *P*. The sequence *w* is now grown as $w\sigma'$.

As IP and PST build tree structures in the learning stage, finding the best suffix involves walking the tree from the root to the node bearing that suffix. The depth of this walk is bounded by the maximum memory length L, but there are cases, in open learning-generating cycles for example, where one does not want to limit *L a-priori*. Furthermore this involves maintaining a particular data structure for *w*, in order to build the candidate suffixes in an efficient way.
A solution might be to use suffix automata instead of trees. In such machines, the current state models automatically the best suffix, so there is no cost in searching it. [Ron96] for instance have proposed Probabilistic Suffix Automata, for which there exists an equivalence theorem with a subclass of PST's. Unfortunately, these are much harder to learn, so they rather propose to learn a PST and transform it afterwards into a PSA. Using this strategy however would invalidate prescription 1.
The best structure we have found so far in order to validate prescriptions 1-4 is the *Factor Oracle* (FO) proposed by M. Crochemore & al [All99]. In the following, we are going to

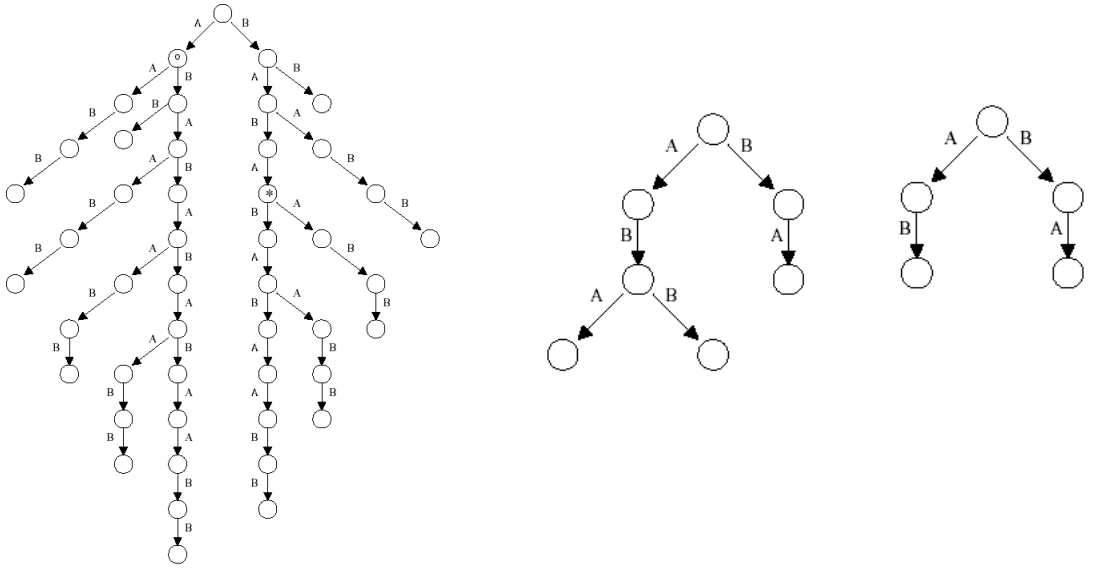

Figure 1*. From left to right, the standard suffix tree, the IP tree and a possible PST learned from the sample sequence w = ABABABABAABB. Probability distributions at each node are not indicated for IP and PST.*

explain how FO fits in the same family than IP and PST, how we use it in learning and generating music, how it conforms to the prescriptions. In the last section, we shall address the question of multi-attribute streams and parallel FO's.

A suffix tree family

Figure 1 shows three models learned from the sample sequence *w* = ABABABABAABB. The suffix tree is the one that carries most information. The implicit memory length can be as big as the sequence itself. The IP model has identified the patterns {A,B,AB,ABA,BA,ABB}, in this order. Although there is a loss of information, due to the shortness of the sample, it has identified that B is often followed by A. The PST has even less information : the leftmost leaves have been removed because ABA has not a significantly better prediction power than BA, and ABB is a singularity.
Obviously, the classical suffix tree (ST) structure serves as a reference structure for the IP and PST representations (the tree representation of an IP or a PST is a subtree of the suffix tree). ST is complete, which means every possible pattern in the sample sequence can be found, and it provides maximum memory length *(|w|-1)*. However, suffix trees are hard to grow incrementally and they are space consuming as they incorporate a lot of redundant information. IP and PST try on the contrary to compute and store the minimal relevant information efficiently. They are actually compression schemes.
We shall be interested now by any learning mechanism which builds incrementally a structure equivalent to a subset of the reference suffix tree, which captures a sufficient amount of statistical information, and is suitable for a real-time generation scheme.
The Factor Oracle has these properties, as we shall show it now.

Factor Oracles

Initially introduced by Crochemore & al in their seminal paper [All99], FO's were initally conceived for optimal string matching, and were extended easily for computing repeated factors in a word and for data compression [Lef00]. Basically, FO is a compact structure which represents at least all the factors in a word *w*. It is an acyclic automaton with an optimal *(m+1)* number of states and is linear in the number of transitions (at most *2m-1* transitions), where *m* = |*w*|. The construction algorithm proposed by the authors is incremental and is *O(m)* in time and space. Here is a brief description of the algorithm :

$w = \sigma_1 \sigma_2 \ ... \ \sigma_m$ is the sample word to be learned. *m+1* states, labeled by numbers from *0* to *m* will be created. The transition function $\delta(i,\sigma_j)=k$ specifies a link from state *i* to state *k>i* with label σ_j . These links run from left to right, if the states are ordered along the sequence *w*. In any FO, the relation $\forall i,\ 0<=i<m,\ \delta(i,\sigma_{i+1})=i+1$ holds.
There is another set of links *S(i) = j*, called Suffix Links, running backward. These links will be discussed further.
w is read from left to right, and for each symbol σ_i the following incremental processing is performed :

```
Create a new state labeled i
Assign a new transition δ(i-1,σi)=i
Iterate on Suffix Links, starting at k=S(i-1), then k=S(k), while k≠⊥
and δ(k,σi)=⊥
      do Assign a new transition δ(k,σi)=i
EndIterate
if k≠⊥ then S(i) = δ(k,σi) else S(i)=0
```

At initialisation, the leftmost state (state 0) is created. Its suffix link is by convention $S(0)=\perp$.
Figure 2 shows the output of the algorithm for *w = ABABABABAABB*.

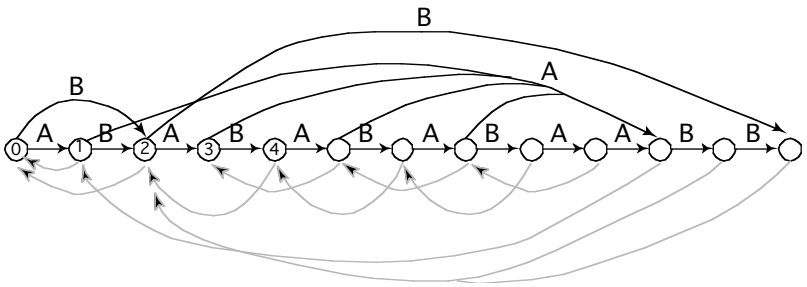

***Figure 2**. The Factor Oracle for w = ABABABABAABB. Black arrows are factor transitions gray arrows are suffix links.*

Turning ST into FO

We want to show that there is a strong connection between suffix trees and factor oracles. We propose a non-optimal algorithm that turns ST(w) into FO(w) for any word $w = \sigma_1 \sigma_2 \ ... \ \sigma_m$.

> Consider the longest path from the root to a leaf in the Suffix Tree. This path bears the string w itself. Number the nodes in this path from 0 to m. A connection from node i to node i+1, 0<=i<=m-1, is labeled by symbol σ_{i+1} . Descend the path starting at root. When there is a bifurcation to another subtree at node i, the longest path starting at this bifurcation and leading to a leaf has to be a suffix $\sigma_j \ ... \ \sigma_m$ of *w* for some *j*.
> Delete the subtree starting at this bifurcation, and add a connection between node i and node j.
> When the transformation is completed, the suffix tree has been « collapsed » along its longest path, and is exactly the factor oracle.

Figure 3 shows the transformation step by step for *w = ABABABABAABB*. Of course, there is loss of information in the process : the fact that whole subtrees are collapsed to a single link between node *i* and *j* may introduce some patterns not present in *w*, and nonetheless recognized by the model. This accouns for the fact that the language recognized by FO (considering all the states are terminal) *includes* all the factors of *w* but is not *equal* to set of factors. This language has not been characterized yet.

Using Suffix Links for generation

Compared to IP and PST, FO is even closer to the reference suffix tree. Its efficiency is close to IP (linear, incremental). It is an automaton, rather than a tree, so it should be easier to handle maximum suffixes in the generation process. In order to show this, some more information on FO must given.

[All99] demonstrates that the suffix link S(i) points to a state j which recognizes the longest suffix in $Prefix_i(w)$ that has at least 2 occurrences in $Prefix_i(w)$. The suffix chain $S^n(i)$ thus connects states where maximal length redundant factors are recognized. To the left of these states, suffixes of these factors will be found (see figure 4).

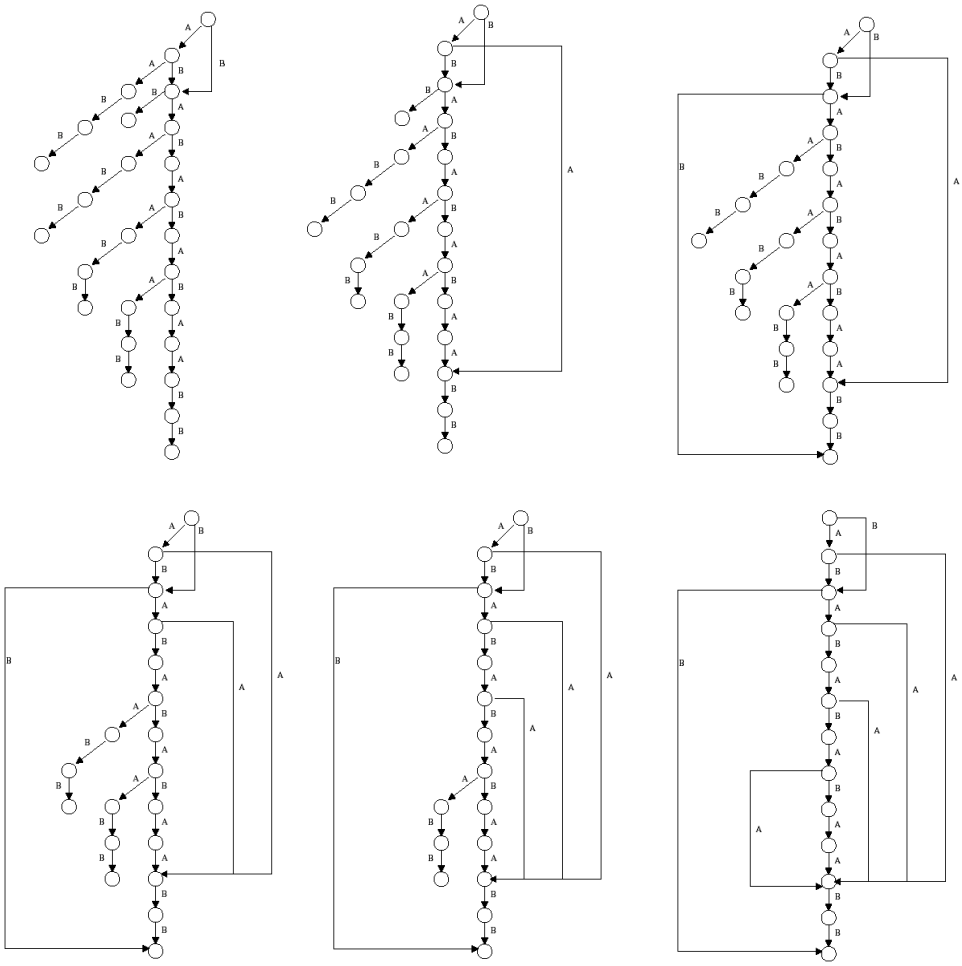

Figure 3. *Turning the Sufix Tree into a Factor Oracle (read left-right,top-bottom).*

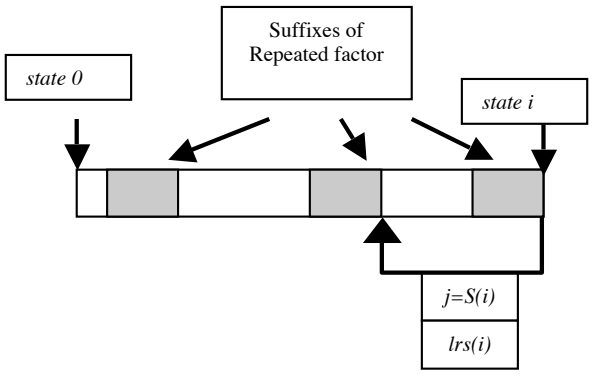

Figure 4. *Suffix links and repeated factors*

Furthermore [Lef00] give a linear method for computing the length *lrs(i)* of the maximal repeated suffix in $Prefix_i(w)$. This gives us the required tool for generating efficiently the next symbol. We suppose the FO has been learned from the sample $w = \sigma_1 \sigma_2 \ldots \sigma_m$. At each generation step, there is a current sequence generated so far $v = \sigma'_1 \sigma'_2 \ldots \sigma'_n$ and an active state *i* in FO such that all the transitions pointing at *i* are labeled by σ'_n.

Incremental generation step :

```
if S(i)=⊥ then q := 1 else q = p
Choose stochastically between 2 options,
   1. with probability q :
      i := i+1
      v := vσ_i
   2. with probability 1-q :
      Choose at random a symbol σ in {σ_j ∈ Σ | δ(S(i),σ_j)≠⊥}
      i := δ(S(i),σ)
      v := vσ
```

The first option means that we are moving linearly forward in the FO thus duplicating a substring of the sample *w*. The second option means we are jumping back along a suffix link. By definition we arrive in a state where a maximal suffix of *v* is recognized. From there, a choice is made among outgoing transitions. These transitions indicate which symbols can possibly follow a suffix of *v*.
The probability variable p controls how close we want to be to the original sample *w*. If it is close to 1, large sections of w will be simply duplicated in the generation of *v*. If it is close to 0, the suffix links will be mostly used, resulting in a bigger rate of bifurcations with regard to *w*.
An interesting option is to consider not only the suffix link starting at *i*, but the whole suffix chain $S^n(i)$, then choose some element in this chain with regard to some criterion. For example, the suffix length *lrs* can be used : choosing a smaller *lrs* will result again in more variety, with smaller factors duplicated from *w*. A probability distribution on the possibles *lrs* might even be used in order to fine tune the variety rate.
A remark should be made on the random choice performed in option 2. of the algorithm : as a difference with IP and PST, there is no probability model in FO, thus there is no probability distribution over Σ attached to each state. Even without a probability model, when we generate long sequences, we should get asymptotically closed to the empirical distribution observed in *w*. However, it should be interesting as a future improvement to add a probability model to FO's.

Multiple channel models

In the case of music there is not only one channel of information as in the text examples seen so far. Rather, several musical attributes must be considered. These attributes describe data or metadata. Data describe the actual musical material and its attribute are: pitch, duration, timbre, intensity, position in bar, etc. Metadata are comments over data that can help the analysis or the generation process. For example, harmonic labels attached to musical data are abstractions, not to be played but to be used as complementary information. Data and metadata can be treated exactly in the same way, so we won't distinguish them anymore.
There are many different ways to arrange musical attributes values: they can flow in parallel information channels, or they may be grouped in a single stream of elements taken in a cross-

alphabet. A cross-alphabet is the cross product of several attribute alphabets (e.g. pitch alphabet, duration alphabet, etc). Different combination of these two solutions may be experimented. It is possible to imagine, for example, a stream of information which elements are drawn from the cross-alphabet built upon harmonic labels and durations. This would make sense in an application where it is considered that a couple (label, duration) is a significant unit of musical information, better to be kept together. Then this stream could be combined with a stream of pitches, resulting actually in a 2-channels information structure.
However, each solution has its drawbacks. Cross-alphabets are simple and compatible with all the models seen so far, but they are more demanding on memory. Multi-channel structures allow different memory lengthes for different attributes, thus optimizing memory, but known models are hard to learn and badly suited to real-time.
Two interesting multi-channel models have been proposed. [Con95] described a *Multiple Viewpoint System*, where a viewpoint is a model based on a cross-alphabet upon a subset of available attributes (e.g. *pitch* x *duration*). In order to predict the next event, independent predictions with respect to each viewpoint (i.e. channel) are combined using a weighted linear combination.
[Tri01] describes a structure called MPSG (Multiattribute Prediction Suffix Graph), which is an extension of PST's where the nodes are labeled not by words in Σ^* but by tuples in $(\Sigma_1^* x \Sigma_2^* x \ldots x \Sigma_n^*)$ where Σ_i is the alphabet for attribute i. One MPSG is build for every attribute i, using at each node a probability distribution that predicts the next value for attribute i with respect to every other attribute. In order to generate the next event, each MPSG is looked for the best multi-suffix and a prediction is for the corresponding attribute, then the attribute values are aggregated to form the next event. It is not clear however if such an event « exists », i.e. has been effectively encountered in the training sample.
In a Factor Oracle, we would proceed differently. First, in FO, there is no reduction of the number of nodes by considering the difference in prediction power between two suffixes differing by a single symbol. The power of FO is to stay close to the completeness of the suffix tree while being simple to compute and efficient in memory size. All the attribute values for a musical event can be kept in a object attached to the corresponding node. The actual information structure is given by the configuration of arrows (forward transitions and suffix links). Multi-channel structures with n channels can thus be simulated by providing n set of typed arrows. A set of arrows for the attribute i will be now characterized by the transition function δ_i and the suffix link function S_i. The n sets can be learned in parallel, with a very simple modification of the learning algorithm.

Let the training sample $W = E_1 E_2 \ldots E_m$ with $E_i \in (\Sigma_1 x \Sigma_2 x \ldots x \Sigma_n)$. W is read from left to right, and for each event $E_i = \sigma_i^1 \sigma_i^2 \ldots \sigma_i^n$ the following incremental processing is performed :

```
Create a new state labeled i
Attach a copy of E_i to state i
For j from 1 to n
      Assign a new transition δ_j(i-1,σ_i^j)=i
      Iterate on Suffix Links,starting at k=S_j(i-1),then k=S_j(k),while
      k≠⊥ and δ_j(k,σ_i^j)=⊥
            do Assign a new transition δ_j(k,σ_i^j)=i
      EndIterate
      if k≠⊥ then S_j(i) = δ_j(k,σ_i^j)
EndFor
```

Musical applications

We have now a rich structure that can give rise to many musical applications. As an example, we propose in this section the description of a possible «real life » machine improviser in a complex performance situation. The machine improviser « listens » to three synchronized sources : a metric source that generates a stream of pulses, a harmonic source that sends harmonic labels, and a melodic source that sends a stream of time-tagged note events. These sources are processes that synchronise in order to align one harmonic label per beat. The granularity of the improviser is the beat : it learns one beat at a time and generates one beat at a time. The three source processes may acquire their data by listening and analysing the signal produced by actual performers, or they can be algorithmic generators, or combinations of both, including combinations that change in time. The improviser continuously learns from the harmonic and melodic source and aligns to the metric source. Upon a triggering signal (such as a human solist who stops playing), it starts (or stops) generating either a melodic part, or a harmonic one, or both.
We give a short specification in the case where it generates both.

Process P_A sends a stream of regular beat pulses p_i, P_B sends synchronously a stream of harmonic labels h_i, P_C sends time-tagged asynchronous midi events m_j. The learning process F is equipped with a factor oracle and receives continuously the messages p_i, h_i, and m_j from P_A, P_B and P_C. As soon as the next beat starts, F performs a learning step on the previously acquired beat p_i, : it collects the set M_i of midi events falling between p_i and p_{i+1}, turns it into a set of descriptors indicating melodic motion, intervalic content, rhythm structure, and codes it into some signature d_i. F creates the next state i in the oracle, associates a copy of M_i to i, and learns h_i and d_i into two separate types of arrow (δ_h,S_h) and (δ_d, S_d).
The generating process F' shares the oracle data structure with F. When it is asked to begin generation, it awakes and waits for the completion of the current beat, p_j then begins its duty. The last state learned in the oracle is j. The main loop of F' is as follows :

```
Loop
   Collect all the states inferred from j by (δd, Sd) into Id
   Collect all the states inferred from j by (δh, Sh) into Ih
   If Id ∩ Ih ≠ ∅
          j ← best inferred state in Id ∩ Ih
   Else
         [with probability p, j ← best inferred state in Id
         [with probability 1-p, j ← best inferred state in Ih
   Send out Mj and hj
EndLoop
```

The states inferred by another state are all the states attainable from the latter either by a δ-arrow or an S^n-arrow. The best inferred state is the one that shares the longest suffix with the originating state. The rationale behind $F'$is : try to generate a combination of harmony and melody which is a well predicted continuation of the harmony/melody that occured in the previous beat ; if you can't, either follow a melodic prediction or a harmonic one, under control of variable p. In any case, the midi events and the harmonic label sent out as a result are consistent because they are associated to the same state (i.e. they were learned together).
In the case we do not want to improvise the harmony *and* the solo, but we would like the improvised solo to be aligned with the incoming harmony, the algorithm is simple : choose among the states inferred by (δ_d, S_d) the ones that have an entering δ_h-arrow labeled with the

same harmonic label than the one currently provided by process P_B (or a compatible label for some harmonic theory). If there'isnt one, a possibility is to remain silent till the next beat and try again, or scan the oracle in order to find a better position.

Conclusion and future works

We have shown the musical potentialities of the factor oracle, a clever data structure that had been mostly demonstrated on textual and biological pattern detection, and we have described the extensions necessary to fit with actual musical situations. We have given a generic architecture for a virtual improviser based on this algorithm in a large class of music, performance situations.
Implementations of the factor oracle and its extensions have been written in the OpenMusic environment [Ass99b] and tested for a great variety of musical styles. The musical prediction power of the oracle has also been compared to human listener prediction capabilities in a set of auditive tests performed by Emilie Poirson in collaboration with Emmanuel Bigand from the LEAD lab, Université de Bourgogne [Poir02]. Nicolas Durand has implemented a set of experiments for pitch/rhythm recombination using parallel oracles [Dur03].
A real time experiment close to the one described in the previous section has been implemented using a communication protocol between OpenMusic and Max, the real time environment invented by Miller Puckette [Puc02]. Marc Chemillier has proposed the real time interaction scheme in Max as well as the harmonic model.
An interesting improvement would be to learn harmonic intervals instead of absolute harmonic labels. In this case, the algorithm would find many more inferred state for a given state, but the midi events in *Mi* would have to be transposed with regards to the context. Interesting combination of transposed patterns would emerge and increase the variety of the improvised material.
As for the oracle itself, experiments have shown that it was fruitful to turn the suffix links into backward *and* forward links, by adding reversed arrows, otherwise the model tends sometimes to get stuck into some region of the automaton.
Finally, a consistent probability model should be proposed for the factor oracle, although it is not clear yet if it would radically change the generation performance.

Audio and video examples demonstrating this work will be installed at :

`http://www.ircam.fr/equipes/repmus/MachineImpro`

Bibliography

[All99] Allauzen C., Crochemore M., Raffinot M., « Factor oracle: a new structure for pattern matching », in *Proceedings of SOFSEM'99, Theory and Practice of Informatics*, J. Pavelka, G. Tel and M. Bartosek ed., Milovy, Czech Republic, Lecture Notes in Computer Science 1725, pp 291-306, Springer-Verlag, Berlin.

[Ass99] G. Assayag, S. Dubnov, and O. Delerue, "Guessing the Composer's Mind: Applying Universal Prediction to Musical Style," *Proc. Int'l Computer Music Conf.*, Int'l Computer Music Assoc., 1999, pp. 496-499.

[Ass99b] G. Assayag et al., "Computer Assisted Composition at Ircam: PatchWork and OpenMusic," *The Computer Music J.*, vol. 23, no. 3, 1999, pp. 59-72.

[Bej01] G. Bejerano and G. Yona, "Variations on Probabilistic Suffix Trees: Statistical Modeling and Prediction of Protein Families," *Bioinformatics*, vol. 17, 2001, pp. 23-43.

[Dub03] S. Dubnov, G. Assayag, O. Lartillot, G. Bejerano, « Using Machine-Learning Methods for Musical

Style Modeling », IEEE Computer, October 2003, Vol. 10, n° 38, p.73-80.
[Dub02] S ; Dubnov, G. Assayag, « Universal Prediction Applied to Stylistic Music Generation » in Mathematics and Music, A Diderot Mathematical Forum, Assayag, Gerard; Feichtinger, Hans-Georg; Rodrigues, Jose Francisco (Eds.), 2002, pp.147-160, Springer-Verlag, Berlin
[Dub98] S. Dubnov, G. Assayag, and R. El-Yaniv, "Universal Classification Applied to Musical Sequences," *Proc. Int'l Computer Music Conf.*, Int'l Computer Music Assoc., 1998, pp. 332-340.
[Dur03] N. Durand, "Apprentissage du style musical et interaction sur deux échelles temporelles", Master's Thesis, Ircam, UPMC Paris 6, Jul 2003. http://www.ircam.fr/equipes/repmus/Rapports/
[Con95] D. Conklin and I. Witten, "Multiple Viewpoint Systems for Music Prediction," *Interface*, vol. 24, 1995, pp. 51-73.
[Con03] D. Conklin, « 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.
[Fed03] M. Feder, N. Merhav, and M. Gutman, "Universal Prediction of Individual Sequences," *IEEE Trans. Information Theory*, vol. 38, 1992, pp. 1258-1270. August 2003
[Lef00] A. Lefebvre, T. Lecroq, Computing repeated factors with a factor oracle, in: L. Brankovic, J. Ryan (Eds.), Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms, Hunter Valley, Australia, 2000, pp. 145-158.
[Pac02] Pachet, "Interacting with a Musical Learning System: The Continuator," *Proc. ICMAI 2002*, Springer-Verlag, 2002, pp. 119-132.
[Poir02] E. Poirson, "Simulations d'improvisations à l'aide d'un automate de facteurs et validation expérimentale", Master's Thesis, Ircam, LEAD U. Bourgogne, UPMC Paris 6, Jul 2002. http://www.ircam.fr/equipes/repmus/Rapports/
[Puc02] M. Puckette, "Max at seventeen." Computer Music Journal 26(4): pp. 31-43.
[Ron96] D. Ron, Y. Singer, and N. Tishby, "The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length," *Machine Learning*, vol. 25, 1996, pp. 117-149.
[Tri01] J.L. Triviño-Rodriguez And R. Morales-Bueno, « Using Multiattribute Prediction Suffix Graphs To Predict And Generate Music », Computer Music Journal, Spring 2001, Volume 25 No. 3, pp. 62-79 .
[Zic87] D. Zicarelli, "M and Jam Factory," *The Computer Music J.*, vol. 11, no. 4, 1987, pp. 13-29.
[Ziv78] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable Rate Coding," *IEEE Trans. Information Theory*, vol. 24, no. 5, 1978, pp. 530-536.