Internship Report

Visualising OMax

Benjamin Lévy

ircam
Centre Pompidou

1st of March - 1st of July 2009

Supervisors : Gérard Assayag
Georges Bloch

PARCOURS MASTER 2
ATiAM
Parcours multi-mentions du Master Sciences et Technologies
Université Pierre et Marie Curie - Paris 6
en collaboration avec TELECOM ParisTech et l'Ircam

Acknowledgement

I would like to thank both of my supervisors Gerard Assayag and Georges Bloch who welcomed me in their office and their OMax. Much appreciations to Gerard's invariable receptiveness and patience and to Georges multitude of sound ideas and kludges.
Many thanks to Karim Haddad for his cheerfulness and to the whole Musical Representation team.
I address a special thank to François Régis who opened my musical views with improvisation many years ago and renewed this experience with OMax during this internship.

One more thank to my parents who supported me over this year of the ATIAM Master's degree.

Résumé

Ce rapport présente une nouvelle visualisation pour le systeme d'improvisation OMax, à base de diagrammes en arc qui permettent de représenter efficacement les répétitions de motifs. Nous étendons cette visualisation pour l'adapter au model interne de connaissance d'OMax : l'oracle des facteurs. Nous developpons de nouveaux outils afin de parcourir et explorer ce graphe et en extraire plus d'information en vue de mieux controler l'improvisateur. Ce nouveau point de vue sur l'oracle des facteurs met à jour une organisation à plus grande echelle dans sa structure. Nous proposons un nouvel algorithme pour extraire ces informations qui s'avèrent musicalement significatives.

Pour cela, nous avons eu besoin de repenser l'intégralité de l'architecture d'OMax et reconcevoir sa structure pour la rendre modulaire et flexible. Tout en reconstruisant des parties entières du logiciel, nous avons maintenu la compatibilité avec les anciennes versions et réimplémenté l'heuristique de navigation existante en en modifiant cependant sa structure pour la rendre cohérente avec ce nouvel environement logiciel : WoMax. Nous avons profité de cette occasion pour perfectionner cette heuristique avec de simple mais efficaces améliorations et nous avons ouvert son architecture pour faciliter de nouvelles expériences.

Mots-clés

Improvisation, Oracle des facteurs, Visualisation, Partitionement incrémentiel, Max/MSP, Musique Assistée par Ordinateur

Abstract

In this work, we present a new visualisation based on *arc diagrams*, an efficient pattern repetition representation, for the improvisation system OMax. We enhance this visualisation and adapt it to the internal knowledge of OMax: Factor Oracle. We develop new tools to browse and explore this graph and extract more information from it to be able to drive the virtual improviser. This new view on Factor Oracle, reveals a higher scale organisation in its structure. We propose a new algorithm to extract this information which is musically significant.

Doing so, we needed to reconsider the whole OMax architecture and redesign its structure to make it modular and flexible. Rebuilding entire parts of the software, we kept a backward compatibility and reimplemented the existing navigation heuristic modifying its structure to make it consistent with our new environment: WoMax. We took this opportunity to refine this heuristic with simple but efficient improvements and open its architecture to ease new experimentations.

Keywords

Improvisation, Factor Oracle, Visualisation, Incremental Clustering, Max/MSP, Computer Music.

Contents

List of Figures

Introduction

Musical improvisation with a partner is based on mutual listening and spontaneous reactions to the others play. How to improvise with a computer is a question the Musical Representation team at IRCAM tries to answer. For several years now, they developed a virtual improviser named OMax which listens to a musician and uses this sound to respond. However, musical improvisation can also require eyesight. Some improvisers need indeed to look while listening and playing to feel differently and communicate with their partners. The look can also reveal or anticipate the partners reactions. How to improvise with an opaque computer ? In this work, we try to make OMax more transparent and reveal its listening to the eyes.

Objectives

OMax project is built on an internal pattern representation, Factor Oracle (FO). Then it improvises with a musician by navigating through this knowledge. The current released version of OMax presents a waveform as the only track of the current state of its knowledge. Apart from parameters of the improvisation logic, its interface just allows to select a particular region on the waveform and define the part of the original recording to use.

In this work, we aim to explore more advanced visualisations of the knowledge contained in FO and consider new interfaces to support both the musician and the sound man controlling the computer-based improvisation. We try several visualisations keeping the most possible the incremental nature of FO.

Enhancing the explicit access to the knowledge embedded in FO opens up new views on this graph. This led us to completely reconsider the OMax architecture and implement a new prototype, WoMax, consistent with the visualisation and more flexible. This new design allows the experimentation of new heuristics and control paradigms.

Having a new overall insight on FO through visulisation, we start to extract higher level information with a new incremental with reinforcement clustering algorithm deeply based on FO properties. This information reveals significant organisation of the musical content corresponding to FO.

Organisation

Following this introduction, the report consists of four chapters:

First we present the background behind our work: after a brief survey on style modelling, FO is studied more deeply to understand the nature of the knowledge imbedded in OMax. Then the current the architecture and functioning of OMax is examined to be able to access significant data. Finally a survey on pattern visualisation in different domains is established.

The second chapter reports the work accomplished during the four months of this internship. We explain and show different visualisations we probed and their pros and cons for different kind of usage. Then a more practical part is laid out with the development necessary to access and display relevant data. Induced by this development, we outline new improvisation strategies with our architecture and representations. Finally, visualisation of the internal knowledge revealed the presence of higher level information. The first version of a new algorithm to extract such information is proposed.

We then summarise and evaluate the work achieved. We lay down in this chapter the current state of development and the visualisations in use. As we regularly submitted our research to tests with musicians, some feedback and remarks from these experimentations are presented. Eventually comments and observation from our personal testing are considered.

This research opened, all along the exploration, novel perspectives around FO and OMax. We list in the last chapter some of these directions and try to estimate their possible interest and followings. Concerning the OMax paradigm, we outline a future environment using the same knowledge (FO). Extracting higher level information may drive to other domains such as Musical Information Retrieval.

Chapter 1

Background

1.1 Style Modelling

"*By Style Modelling, we imply building a computational representation of the musical surface that captures important stylistic features hidden in the way patterns of rhythm, melody, harmony and polyphonic relationships are interleaved and recombined in a redundant fashion.*" [11]. In other words, style modelling is the generation of new musical material respecting implicit rules learnt from another musical material (often referred as the source). Whether the system infers the implicit rules of the source or mimics the surface without trying to guess these rules depends on the representation model used.

In this field, investigations have been done with dictionary based models such as LZ compression algorithm (Incremental Parsing (IP) methods) or Prediction Suffix Trees as reviewed in [12]. Both methods build a tree encoding patterns occurrences and allows to calculate probabilities on the continuation of a given pattern. A rather large review of music generation methods is given [9].

In the years 2001-2002 [19] the Music Representation team in IRCAM started to use a novel representation of pattern repetition, Factor Oracle, coming from the genetic research on patterns in genome. Since then, they studied and successfully used this representation for style modelling and music generation and built a real-time improvising system called OMax. The internal knowledge of this system was developed thanks to OpenMusic, a high level environment programmed in LISP. However, due to the complexity of the model and the difficulty to represent entangled structures, there is no explicit visualisation in their system.

1.2 Factor Oracle

Factor Oracle (FO) was introduced by Allauzen & al. [1, 2] to optimally compute the factors in a string of characters — i.e. the substrings contained in the string. This automaton is acyclic and recognises at least all the factors of the string. It has $m + 1$ states, where m is the length of the string, and at most $2m + 1$ transitions. FO of the string `abaabacba` is presented Figure 1.1 graphed with the **Graphviz** package [16] embedding the **.DOT** description language (which we used as the first step towards FO visualisation). FO is constituted of two types of links:

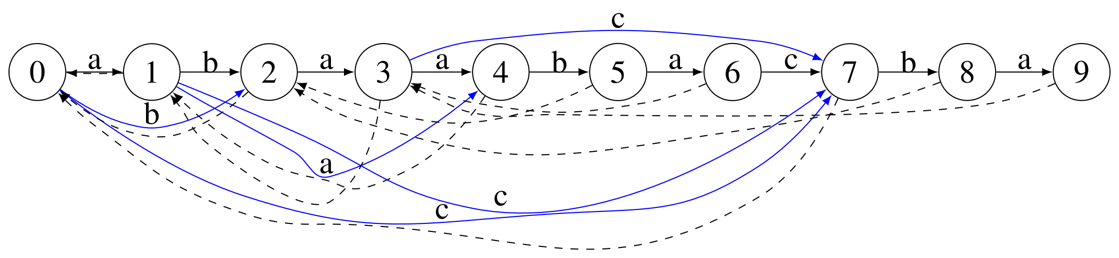


Figure 1.1: Factor Oracle of the string `abaabacba`

- forward links called *transitions* represented by the *transition function* $\delta(j, \sigma_k) = k$ where j is the origin state and k the destination state of the link **with** $\boldsymbol{j < k}$ (plain links on Fig. 1.1). σ_k represents the symbol (character, musical descriptor or any data FO graphs) corresponding to state k.

- backward links called *suffix link* represented by the *suffix function* $S(j) = k$ where j is the origin state and k the destination state of the link **with** $\boldsymbol{j > k}$ (dashed links on Fig. 1.1).

The first state, numbered 0 has no associated symbol and by convention $S(0) = -1$.

Once built, FO can be divided in three parts depending on which type of links are considered:

- The graph *skeleton* constituted by every state and the transition to its direct following:

$$\delta(i, \sigma_{i+1}) = i + 1 \quad \forall i \in [0, m - 1]$$

  represents exactly the input sequence (black plain links on Fig. 1.1).

- Every transitions except those of the skeleton, can be seen as forward jumps in the automaton (blue links on Fig 1.1). Stepping through the automaton using these transitions allows to generate all the factors of the original sequence and a some more... Characterising the exact language a FO recognises, remains an open problem nowadays.

- Suffix links connect pattern with the first occurrence of the same pattern (dashed links on Fig. 1.1). When a new symbol occurs, the suffix link goes to state 0 (for example, on Fig. 1.1, the appearance of the letter `c` in the string `abaabacba` creates a suffix link from state 7 to state 0).

1.2.1 As an automaton

Two construction algorithms were proposed in the original paper [2], one of them is incremental and linear in space and time ($O(m)$) which makes this automaton very efficient to represent "on-line" music sequences. The incremental construction algorithm is the following:

Assuming the automaton is built to state $i-1$ and we want to add the state i corresponding to the symbol σ_i

```
**Create** a new state numbered $i$
**Assign** a new transition labelled $\sigma_i$ from state $(i-1)$ to the new state:
    $\delta(i-1, \sigma_i) = i$
**Iteratively backtrack** suffix links from state $i-1$:  $k = S(i-1)$ **doing**
    **if** no transition from state $k$ is labelled by the same symbol as $\sigma_i$
        **then Assign** a new transition from state k to the new state:  $\delta(k, \sigma_i) = i$
            and **Track** the suffix link:  $k = S(k)$
        **else End backtrack**
**if** backtrack ended before state 0:  $k > -1$
    **then** an existing pattern is recognise and linked:  $S(i) = \delta(k, \sigma_i)$
    **else** the suffix link goes to state 0:  $S(i) = 0$
```

Figure 1.2 shows as an example, the construction of Factor Oracle of the string abaababacba.

Lefebvre, A. and Lecroq, T. added [17] shortly after the original paper a light modification to the construction algorithm to compute the length of every suffix link in FO. Their algorithm uses (in the function LengthCommonSuffix) the same backtracking mechanism in the suffix path to find the first common suffix between the last built state and the first occurrence of the prefix of the pattern created with the state to add. They added to FO structure a new data called *Length of Repeated Suffix* ($lrs(i)$) representing this data and published an new incremental algorithm preserving the linear complexity of the construction. Here is the new algorithm with highlighting on modifications:

Assuming the automaton is built to state $i-1$ (including lrs) and we want to add the state i corresponding to the character σ_i

```
  **Create** a new state numbered $i$
**Assign** a new transition labelled $\sigma_i$ from state $(i-1)$ to the new state:
    $\delta(i-1, \sigma_i) = i$
**Retain** state $i-1$:  $\pi_1 = i-1$
**Iteratively backtrack** suffix links from state $i-1$:  $k = S(i-1)$ **doing**
    **if** no transition from state $k$ is labelled by $\sigma_i$
        **then Assign** a new transition from state k to the new state:  $\delta(k, \sigma_i) = i$
            **Retain** this suffix:  $\pi_1 = k$
            and **Track** its suffix link:  $k = S(k)$
        **else End backtrack**
**if** backtrack ended before state 0:  $k > -1$
    **then** an existing pattern is recognise and linked:  $S(i) = \delta(k, \sigma_i)$
        and $lrs(i) =$ **LengthCommonSuffix**$(\pi_1, S(i) - 1) + 1$
    **else** the suffix link goes to state 0:  $S(i) = 0$
        and $lrs(i) = 0$

**LengthCommonSuffix**$(\pi_1, \pi_2) =$
    **if** $\pi_2$ is the suffix of $\pi_1$:  $\pi_2 = S(\pi_1)$
        **then Return** $lrs(\pi_1)$
    **else While** no common suffix is found:  $S(\pi_2) \neq S(\pi_1)$
        **Backtrack** suffix links from $\pi_2$:  $\pi_2 = S(\pi_2)$
    **Return** the least $lrs$ before common suffix:  $min(lrs(\pi_1), lrs(\pi_2))$
```

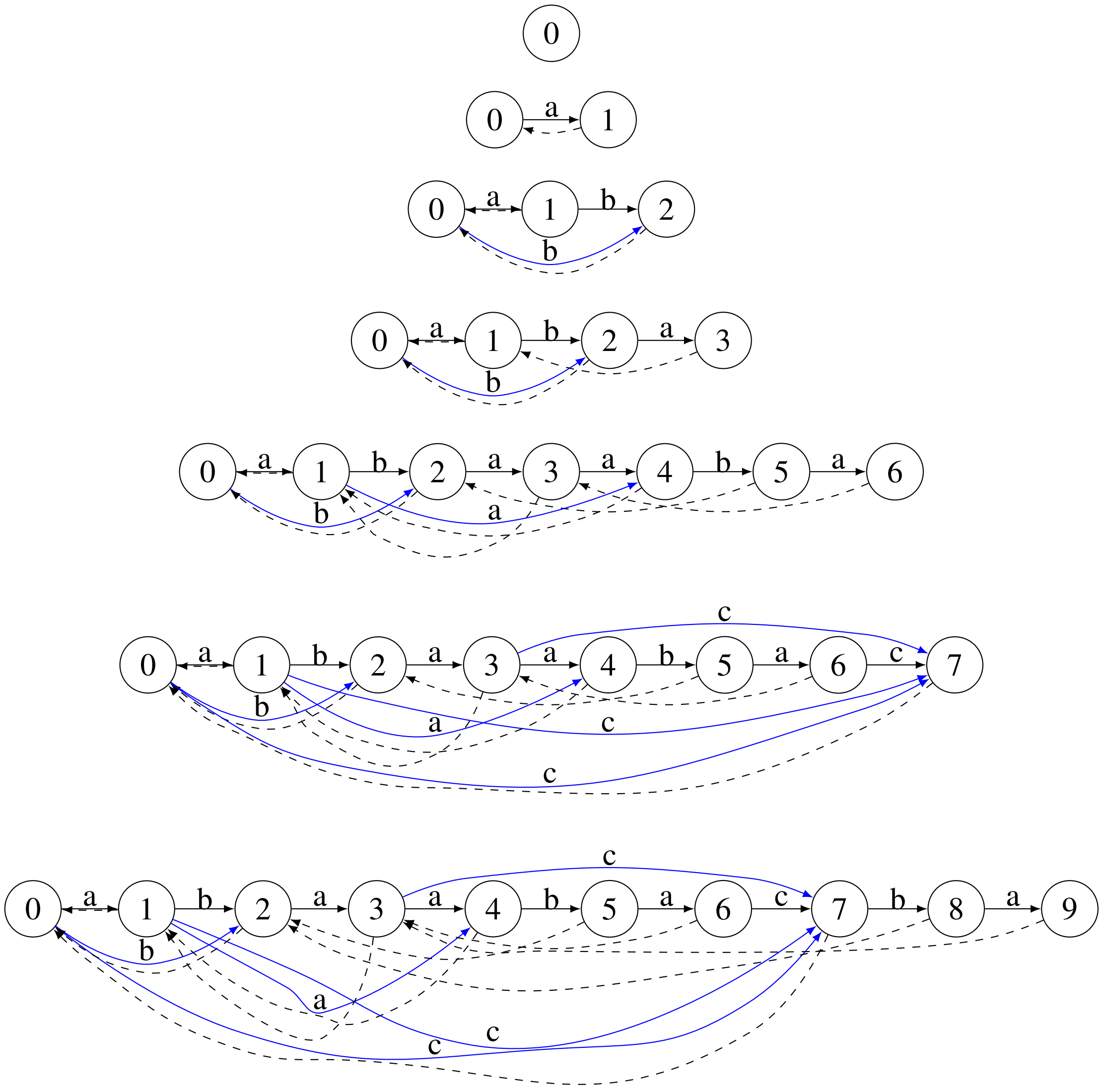

0
0 a 1
0 a 1 b 2 b
0 a 1 b 2 a 3 b
0 a 1 b 2 a 3 a 4 b 5 a 6 b a
0 a 1 b 2 a 3 a 4 b 5 a 6 c 7 b a c c c
0 a 1 b 2 a 3 a 4 b 5 a 6 c 7 b 8 a 9 b a c c c

Figure 1.2: Incremental FO construction of string abaabacba

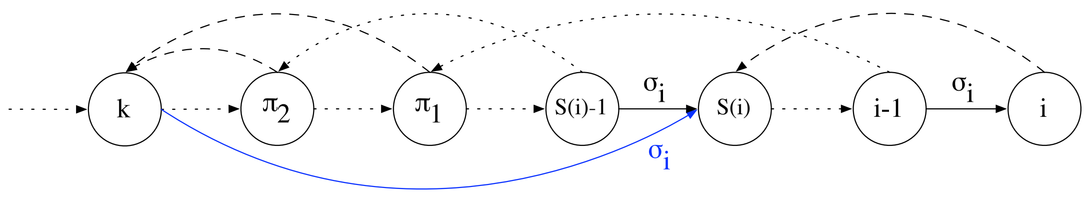


Figure 1.3: Computation of *lrs*. Dotted lines represent series of transitions or suffix links

1.2.2 As a tree

FO extended with *lrs* is an automaton recognising at least all the factors of the word it represents. In [6] Assayag, G. and Dubnov, S. give an interesting comparison between FO, PST and IP. They demonstrate and show that the suffix tree of a word can be turned into FO of the word.

Another way to look at FO is to consider only the suffix links. Reading backwards these links creates a tree presented Figure 1.4 which root is state 0. In this *suffix link tree* (*SLT*) each node shares with its father a common suffix of the length indicated by the *lrs* (label of the link on Fig 1.3). Two sons of a node (except the root) share a common suffix but the length of the suffix each one shares with the parent can be different, depending on the *lrs*. Therefore, the common suffix they share is given by the minimum of both *lrs* as previously presented 1.2.1 in the second algorithm. The computation of a common suffix can then be considered as the search of a common ancestor in the suffix link tree (cf. Fig. 1.5).

1.2.3 As forest of trees

However, suffixes of length 0 are not interesting nor are, most of the time, suffixes of small length — depending on the descriptor used, states in musical FO can last from $5ms$ (spectral descriptors) to more than a second (pitch). The SLT can be filtered (pruned) to remove suffix links with *lrs* below a given threshold. This transforms the tree into a forest of separated non-overlapping sub-trees (SLTs) of the whole SLT (see for more illustrations [3]). The number and size of these SLTs depends on the minimum *lrs* considered and on the original input sequence. An example is given Figure 1.6 with a threshold of 2. As we can see, all the nodes in a pruned SLT share a common suffix of length superior or equal to the pruning threshold.

On the contrary of PST, in pruned SLTs, every node is an effective occurrence of the least common suffix. Therefore, the forest of SLTs graphs all the patterns of the pruning length contained in the input sequence and all the occurrences of these patterns. This is the very basis of OMax project .

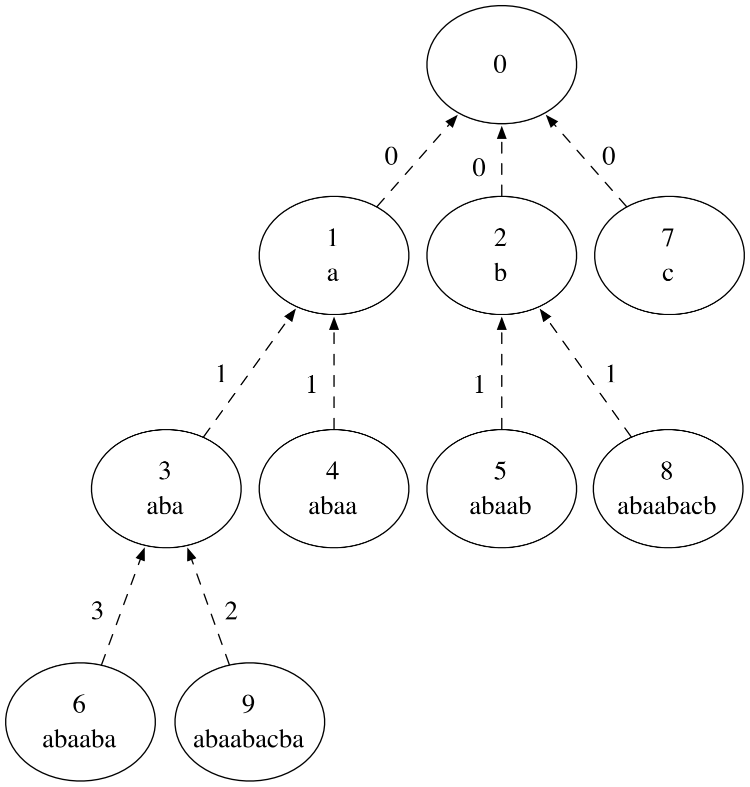


Figure 1.4: Tree of the suffix links in FO of `abaabacba`, suffix links are labelled with *lrs*

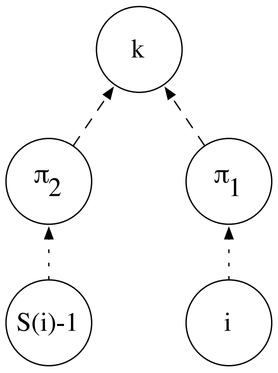


Figure 1.5: Tree representation of the common suffix. Dotted lines are series of suffix links

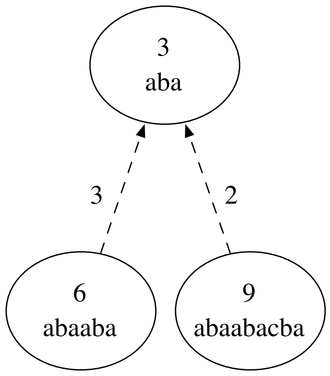


Figure 1.6: With a pruning *lrs* of 2, the forest of SLTs of `abaabacba` is reduced to a sole tree

1.3 OMax

Started 2004 after some other attempts at style modelling [11] using PST and IP representations, OMax project uses FO for both analysis and generation stages of its improvisation system. At first it encoded a representation of music with MIDI data, then an audio layer (named Ofon [4]) was added to automatically extract pitch information from an audio stream. Lately OMax has been also adapted to use spectral descriptors as Mel Frequency Cepstral Coefficients or Linear Prediction Coefficients to build its knowledge [8].

1.3.1 Architecture

OMax uses two computer music environments [5]: OpenMusic and Max/MSP. Max/ MSP, dedicated to real-time processing, is used for detection and segmentation of the input audio stream. It records this streams in a buffer and sends segmentation data to OpenMusic through Open Sound Control (OSC) protocol. OpenMusic, specialised in higher level operations builds and browses the knowledge model (FO) to send back to Max/MSP the improvisation data (results of the knowledge browsing, see 1.3.2). The latter schedules and renders the improvisation to output an audio stream. See Figure 1.7 for an overall diagram of this architecture. Learning and improvising are parallel processes and can take place independently and concurrently.

In OMax, one state of FO is constituted by a date and a duration over which descriptors data are considered as stable. It can be a note when monophonic pitch is used or a chord with polyphonic slices but also a particular timbre with spectral descriptors.

1.3.2 Improvising with FO

As explained in 1.2.3, a pruned SLT references every occurrence of a pattern contained in the input sequence. The heuristic developed in OMax [3] is based on this property. After reading a sequence of consecutive states — the number of these is chosen with a parameter called *continuity* — the improviser searches for other occurrences of the pattern it just read, by computing the pruned SLT containing the last read state. Every state in this tree is a candidate to jump to because they all share

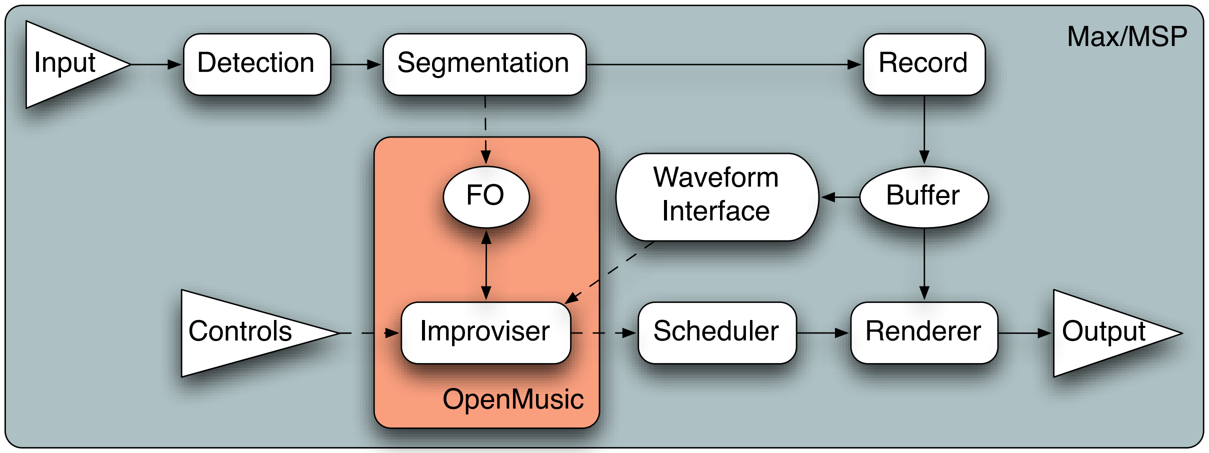


Figure 1.7: OMax architecture. Dashed lines denote usage of OSC protocol

a common suffix (called *context*). This creates a musical continuity. However, nodes in the tree sharing a longer context are obviously musically more interesting. An efficient navigation scheme in SLTs is developed in [3].

Additional functions are implemented in the improviser to create a better musical sequence. A taboo queue is filled along the navigation with the effective jumps to avoid exact loops and a rhythm coefficient is calculated to ensure a musically consistent sequence. This coefficient compares the duration of the pattern occurrence between the origin of the jump and a the destination (see 2.3.1). Better (i.e. smaller) coefficients are more likely picked among the selected solutions — the 6 solutions with the longest contexts.

1.3.3 Interface

From the OpenMusic part of OMax, nothing is actually viewable except OSC messages. However, A.R.J François et al. used OMax principles to create “*a multi-modal interactive musical improvisation system that explores the role of visual feedback in performer-machine interaction*” [13, 14]. In this system named MIMI, they chose to separate the learning stage from the improvisation stage. In a first phase, the system is prepared with a musical material. Then in a second phase (eventually repeated) the musician improvises with the computer. They use the same visualisation paradigms for both stage (Figure 1.8). The representation is based on two piano roll visuals showing notes on the vertical axis and time on the horizontal axis. The first piano roll (lower half) displays the learnt sequence and the current location of the improviser. The second piano roll (upper half) is used for the improvisation stage and represents both the human new improvisation (in red) and the computer based generation (in blue). This latter is planned ahead so the future is also visualised.

Displaying the future of the computer based improvisation is very interesting because it allows the musician to anticipate the machine notes and adapt his own improvisation. However, this system works with MIDI data only and its visualisation requires to separate the analysis stage from the improvisation stage.

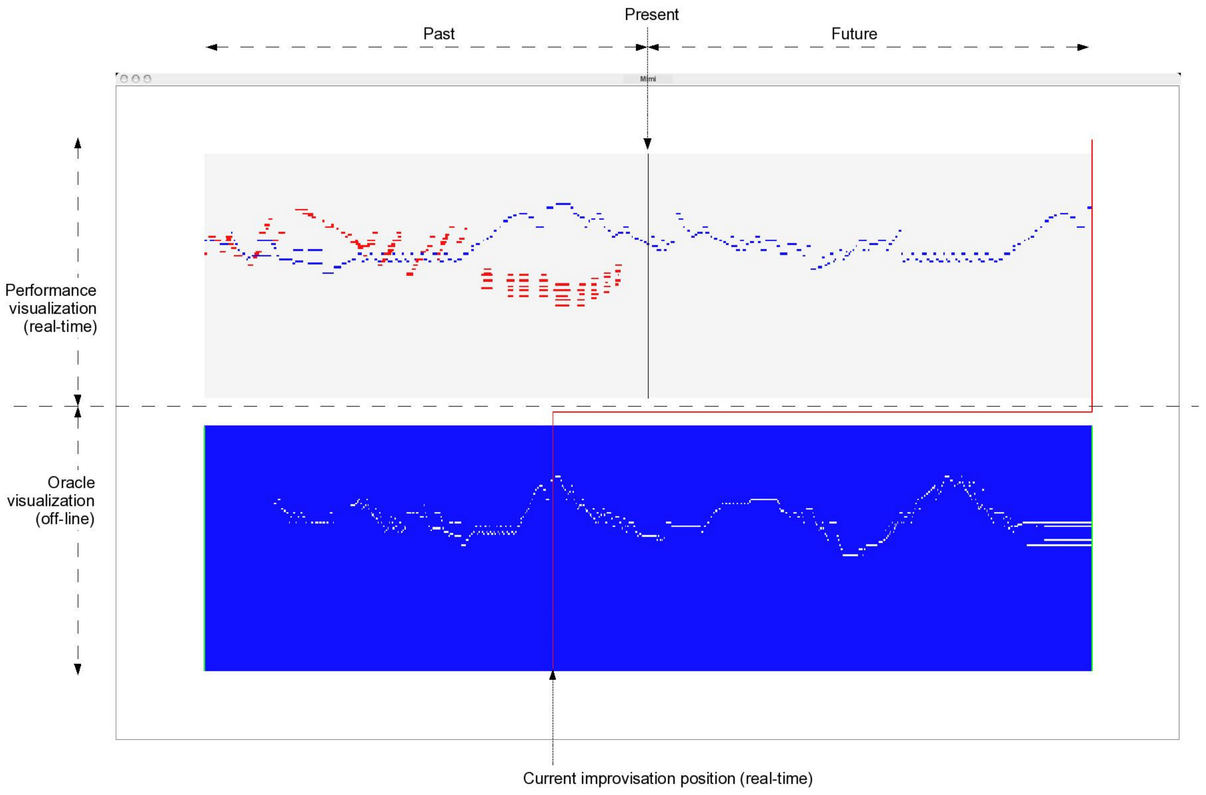

Present
Past
Future
Mimi
Performance visualization (real-time)
Oracle visualization (off-line)
Current improvisation position (real-time)

Figure 1.8: MIMI interface [13]

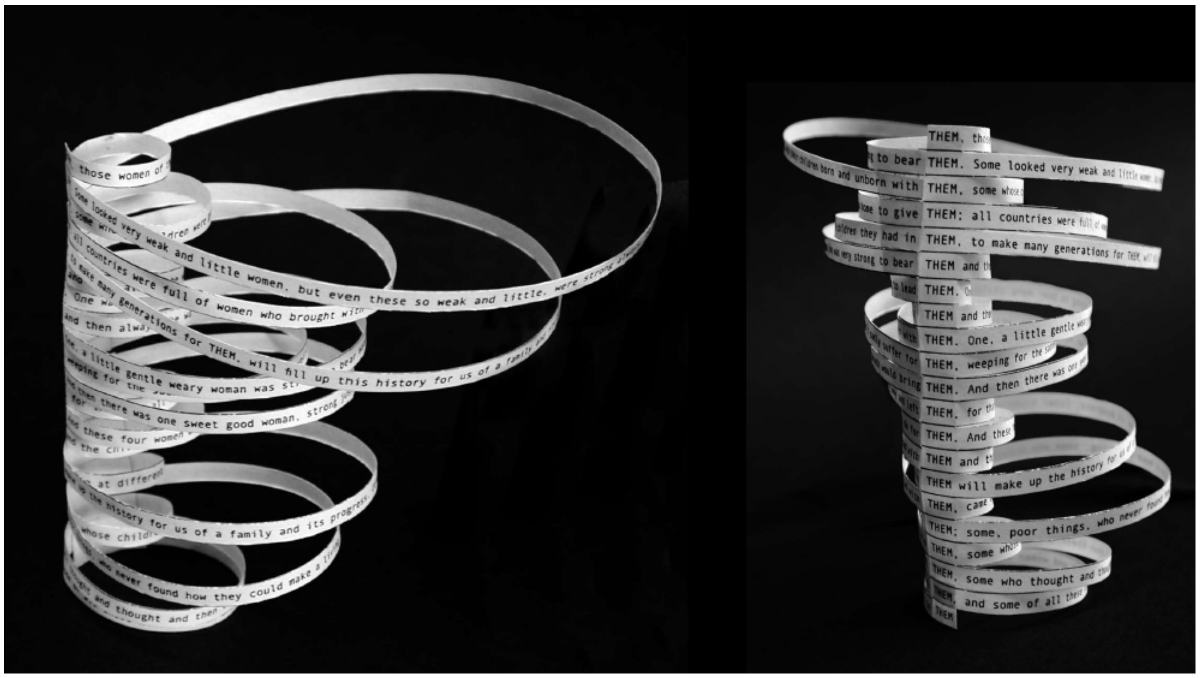


Figure 1.9: 3D repetition graph: "loops of text"

1.4 Pattern Visualisation

Visualising repeated patterns in a sequence is a complex problem because of the profusion of these patterns as soon as the sequence gets long. It creates a tangled abundance of references from one pattern to its repetitions which makes very difficult to have a clear overview.

1.4.1 Text

This problem is encountered in text analysing and exploring. Many research and developments have been done about data mining in texts. However few representations profit from this field as explained in [21]. Some exploration tools are presented in [10] and evaluated on the very studied (at least in the data mining and pattern recognition communities) *Making of Americans* from Gertrude Stein. They provide an exploration interface which may suit to text but would be difficult to adapt to music because of the timed nature of music. Indeed, divided in many different parts, it gives precise and useful informations (context, section, most frequent patterns . . .) but no linear overview (see 1.4.3).

Other interesting visualisations of patterns in texts are presented in [20]. A particularly nice and pertinent one is what they call *repetition graph*. "*Repetition graph is an attempt to show all repetitions of all words in an entire text*". It preserves the text order and gives a visual notion of distance between word repetitions. Nevertheless, it is quite difficult to display with long texts except when focusing on one particular word and "rolling up" the graph to create a 3D visualisation as shown Figure 1.9.

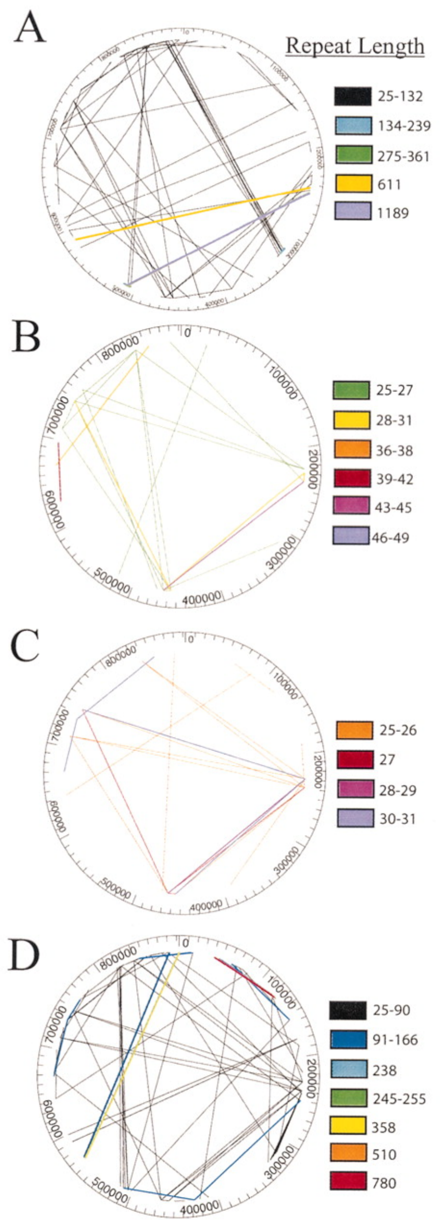


Figure 1.10: REPvis component of REPuter: a tool to analyse repetitions in genomic sequences.

1.4.2 Genetics

Another field where pattern repetition is a whole research subject and a real issue is genetics. Actually the first usage of FO given in [17] is the detection of long repetition in long genomic sequences. It is developed by [15] and given a visualisation interface: *REPvis*. Part of the *REPuter* family, this software allows to compute and display the longest pattern repetition in a genomic with an indication of their length. For such an application, the whole input sequence is defined at once and does not change in time which allows static visualisations like circle graph (Figure 1.10) but exclude an "on-line" representation as needed for Music.

1.4.3 Music

The timed character of music is a very strong constraint to efficiently represent an "on-line" analysis. To be understandable a musical representation seems to require a time-oriented visualisation indeed linear. Otherwise an ever evolving representation becomes difficult to browse and especially to understand at once.

A visualisation of repetitions named *arc diagrams* is developed in [23] and tested on texts, music pieces, sequences of bits, computer programs . . . It keeps the (time) order of the input sequence and gives very promising results on music (the associated artwork is presented online [22] as a Java applet). The idea is to represent each repetition of a pattern with an translucent arc which width correspond to its length. The ends of this arc indicat two consecutive occurrences.

However, the implementation of this work is based on a suffix tree and works with MIDI files which exclude a direct usage in OMax real-time context. But we will adapt this idea to our needs and go further in the use of it for music structure representation.

Chapter 2

Work

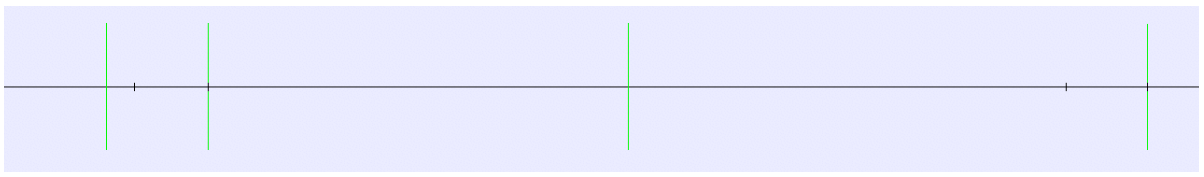
Past the bibliographic part presented chapter 1, the main idea which drove our work was to make available to our eyes the very powerful knowledge embedded in OMax. The audio results of OMax improvisation had already been evaluated and refined along the years but no representation of the internal knowledge was used or displayed except the waveform of the audio recording. Beyond the interface itself, a visualisation of this knowledge is the starting point of new views on FO for a better understanding, new interactions and higher level information extraction.

Rather than describing the chronologic course of the work, we will first present the representation part of our work with the different visualisations we came to. Then we will explain the development which were required and the architecture design necessary to build these representation. We implemented in this architecture the improvisation heuristic already in OMax but doing so, we redesigned the improviser to add new possibilities. We will show which improvements were made and how it can change the way of interacting with the system. Finally, our visualisation opened higher information extracting opportunities. We present a first high level clustering algorithm based on FO.

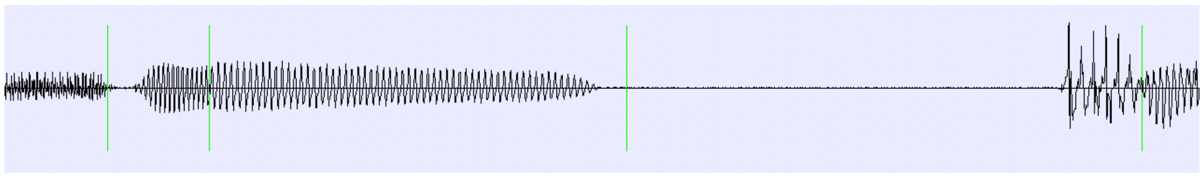
2.1 Visualisation

The first idea when thinking of FO representation is, naturally, to show the graph described by the algorithm (cf. 1.2.1). As mentioned 1.2 and shown Fig. 1.2 the first representation we came across was the **.DOT** [16] description format which allows to incrementally display the whole construction process. However, there is currently no software capable of rendering graphs described in this format with more than 200 or 300 states — A musical sequence, especially described with spectral descriptors can correspond to an FO of more than 10,000 states. The interface and interaction possibilities of this format are also very limited.

As the improvising heuristic [3] doesn't use the transitions (forward links, see 1.2) of the graph but only the suffix links, it was judicious to drop these links in the visualisation instead of trying to represent the whole FO. Nevertheless the following of states in a musical sequence is an essential information often represented as a timeline (succession of staves on a music sheet, linear timeline in most of the softwares). We kept this as a constraint and eliminate graph representations which



(a) without waveform. The green bars are Note Off



(b) with waveform. Green bars are Note Off

Figure 2.1: Detail of the timeline in the case of monophonic pitch descriptor

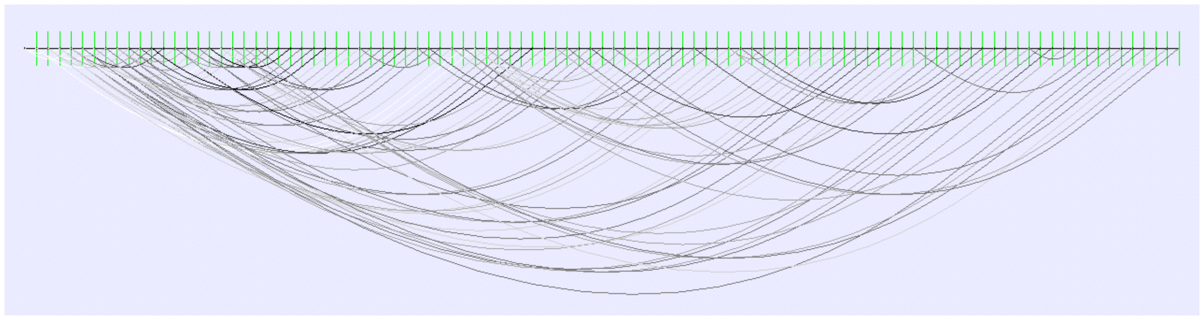


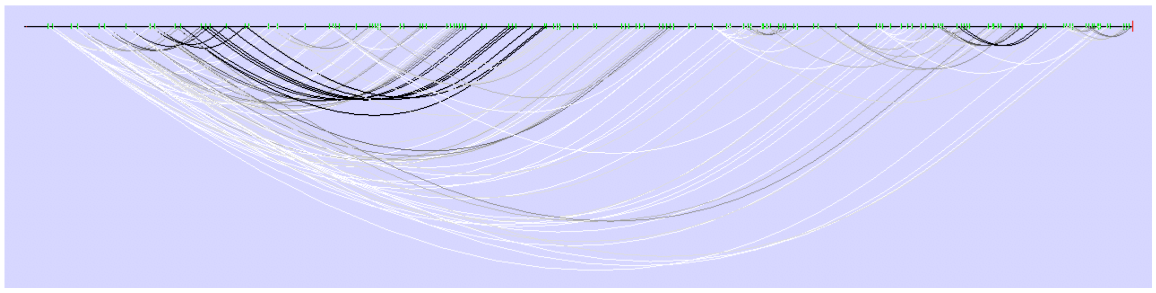
Figure 2.2: Basic FO representation of a 100 letters (a, b or c) sequence. The grey scale denotes the context lengths from 0 (white) to 6 (black)

make use of dimensional transformations such as hyperbolic spaces [18] as it would be too far from a usual music representation. Linear timeline allows us also to include in our new visualisation what was the only representation in OMax: the waveform (see an example Fig. 2.1).

2.1.1 Analysis Stage

As pertinently noticed in [14], there are two different parts to visualise in a system like OMax: the analysis or learning stage to inform of the current state of knowledge and the improvisation stage which browses the knowledge to generate new sequences.

Pruning FO from its transitions and displaying it directly in Jitter (the video/visual part of Max/ MSP), we obtained an incremental visualisation of the rough knowledge (FO) contained in OMax. Figure 2.2 gives an example of this display for a sequence constituted of a hundred letters in the alphabet a, b and c. The skeleton (see 1.2) is drawn with an horizontal (time)line (from left to right) and vertical green bars show the states of FO. Suffix links are graphed underneath with simple bezier curves on a grey scale to visualise the context length (lrs). By convention, for letters, the "duration" is 1 and the left to right order is the reading order of the sequence.


(a) with all suffix links with context length from 1 (white) to 13 (black)

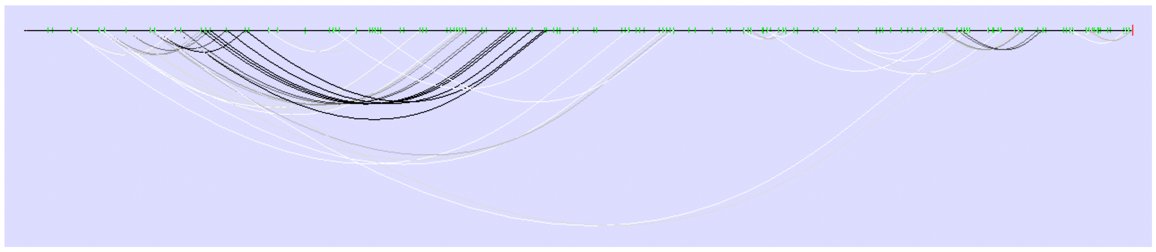

(b) pruned to keep links with context length superior to 3 (white)

Figure 2.3: Basic FO representation of 60 seconds of a saxophone melody

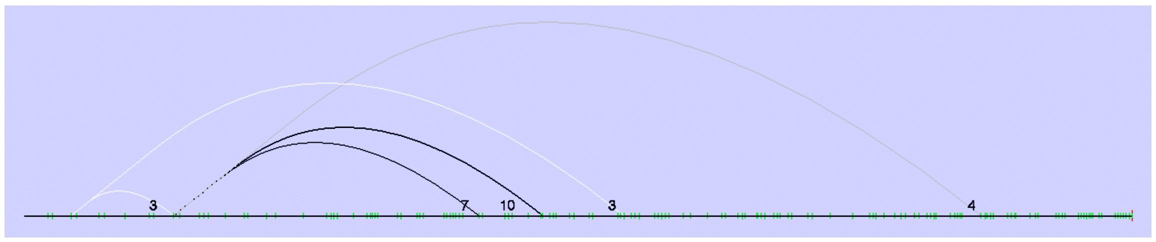


Figure 2.4: Visualisation of a particular SLT with minimal context set to 3 (white)

Although when displaying letters sequences we could have shown the letters corresponding to each state, in the case of musical descriptors there is no point in indicating the MIDI pitch as a number or all the more, the 5 or 6 floating point coefficients of a spectral descriptor. Figure 2.3 shows the FO of 60 seconds of a saxophone melody with monophonic MIDI pitch segmentation (this will be the descriptor used further on in this report).

Once this first visualisation in place, we tried to highlight what FO and suffix links basically encode i.e. pattern repetitions in the sequence. As we have seen before (1.2.3), the last state of every occurrence of a particular pattern is a node in the pruned SLT containing all of them. We can then access this information by computing on demand the SLT from a particular state and visualising it. This is presented Figure 2.4 with display of *lrs* of the links in the tree. In this representation, every end point of a link is the end of the last note of the common pattern (which has 3 notes). We can directly see that some occurrence of this pattern share a much longer context (suffix). For example the two end points with *lrs* of 7 and 10 are obviously two sons of a same

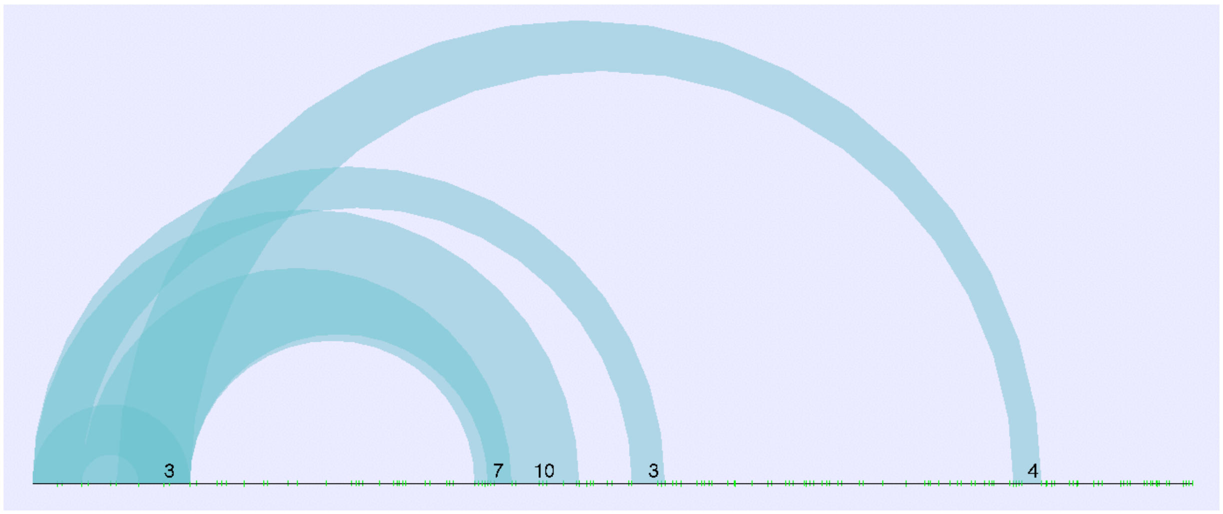


Figure 2.5: Visualisation of a particular SLT with arcs

parent in the SLT, thus they share a context of 7 but they do not share such a long context with the third son of this node which has an *lrs* of 4 only.

These simple curves already reveal a part of the knowledge OMax uses. It allows us to localise in the past occurrences of a pattern and read the length of it. However, it is not immediate to read and understand. Therefore, we implement the visualisation described in [23]: *arc diagrams*. One of the main advantage of large arches is that the thickness of the arch, representing the duration of the pattern, encodes two data which are actually linked and musically relevant : the length of the pattern in terms of states (or notes in the case of monophonic pitch) and the actual time duration in milliseconds or seconds. Combining these two pieces of information in one visual object is meaningful and original compared to what is done on MIDI files in [23]. Figure 2.5 shows the same SLT as Figure 2.4 with this new visual. Apart from the relation between the thickness of arches and duration of the patterns, this representation shows also a new piece of information which was difficult to extract from the simple links: overlapping patterns. With the translucency of the colour, we easily notice that the same two occurrences previously mentioned which share a common suffix of 7 notes are actually overlapped, the three last notes of the first one being the three first notes of the second one.

Freed of the grey scale because of the thickness of arches, we went further than [23] by using the color to encode the pattern. With the use of different colours for different SLTs, we can visualise several STLs simultaneously as presented Figure 2.6. Unfortunately, finding a relevant meaning of the colour (except differentiation of the patterns) is a complex problem we had not enough time to solve.

Yet, drifted from the arc diagrams, we designed another visualisation which could be called *triangle diagrams*. Even though the idea of replacing the arcs with triangles is very simple, the results makes obvious a very musically interesting parameter: the time stretching of patterns. Actually, the difference of thickness between the to end points of an arch was already showing this information but triangles emphasise its impact and make it flagrant as we can see on Figure 2.7.

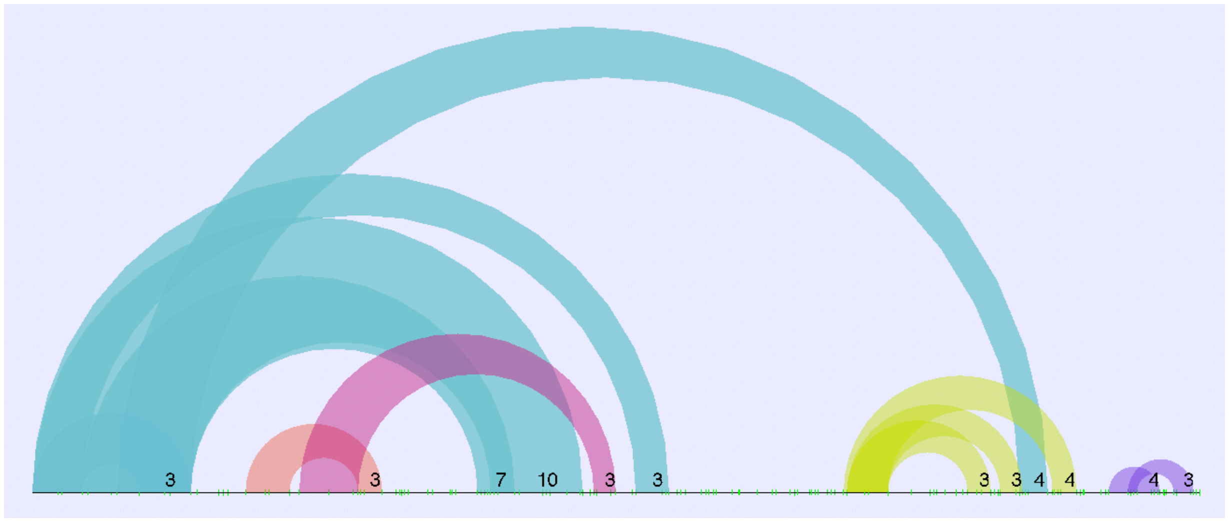


Figure 2.6: Several SLTs graphed with different colors

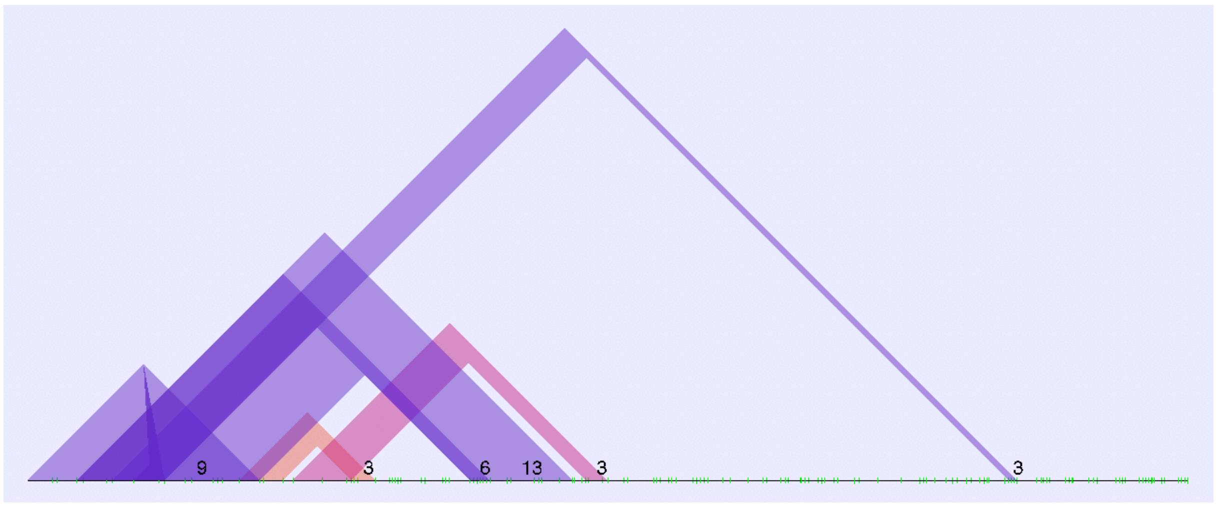


Figure 2.7: Several SLTs displayed with triangles. The difference of thickness between the two branches of a triangle reveals different duration of the same pattern

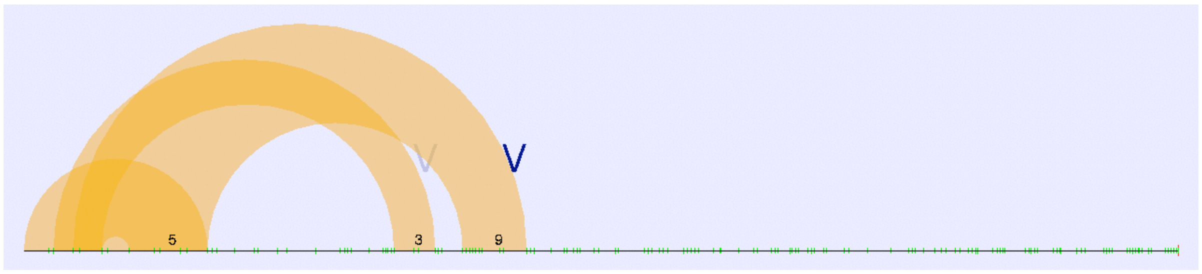


Figure 2.8: Visualisation of a jump during the generation stage. The dark arrow indicates the current state, the light arrow indicates the destination of the jump. Arches denotes the path in the SLT from one state to the other, common context is the minimum of encountered lrs: 3

All these visualisations are available on demand and therefore can be rendered to explore FO with the mouse, to display selected SLTs or trace all the SLTs together but also to show in real time incoming information from data stream — as soon as it is stored in FO. The "on-line" representation allows to see at once the repetitions of the pattern finishing by the last played note, the time localisation of past occurrences of this pattern and some differences in tempo (or time stretch) if any. Even though good musicians do have conscience of this kind of information — along with much more —, this rather simple use of FO is a very accurate knowledge.

2.1.2 Improvisation Stage

As explained in 1.3.2 the improvisation part of OMax uses FO to read consecutive states then jump to another node in the SLT containing the pattern it just read. We can easily distinguish two part in this process corresponding to two different information to show. The first part we are interested in visualising is jumps. We use the previously presented visual to show localisation of both the origin and the destination of the jump, but also the path in the SLT which links them and the context they share. Figure 2.8 shows an example a jump ; looking at the lrs at the bottom of arches, we can deduce the common context between the origin and the target of the jump.

The second part consists of the contiguous states read and by extension, the past of the improvisation which will be constituted of fragments of the original sequence. Visualising this on the original timeline hides the chopping nature of the improvisation algorithm. On the other hand, representing the improvisation with a second linear timeline collapses the jumps. Therefore we decided to represent fragments with the same visual as the original sequence but floating above it and aligned with their original position. The order in the fragments is given by following the suffix paths of each jumps. Figure 2.9 gives an example of this representation.

Combining the visualisations both mechanisms of OMax improvisation gives an interesting but quite dense representation as shown Figure 2.10, especially when the improvisation grows longer and many jumps are added. But a short term visualisation — the last 4 or 5 jumps and their continuations for example — is a good balance for a briefly trained person.

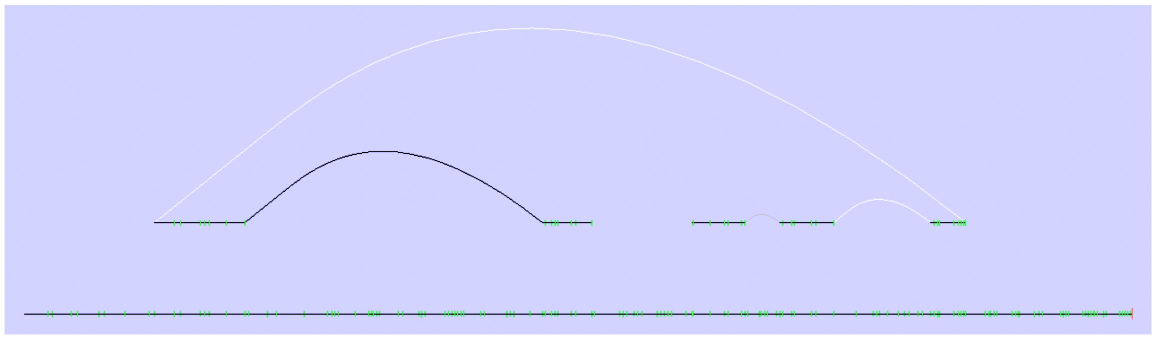


Figure 2.9: A short improvisation on the original sequence (timeline at the bottom). Follow the links to read the order of the fragments

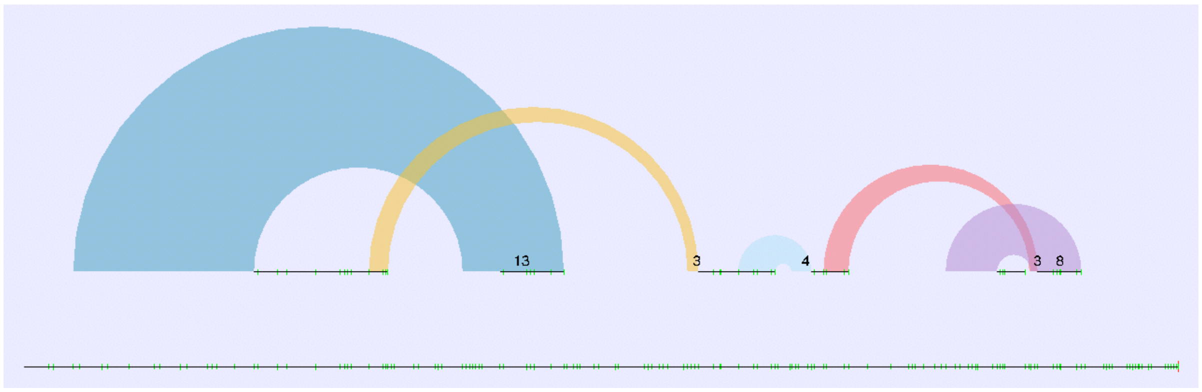


Figure 2.10: Another short improvisation visualised with both fragments and arches visuals.

Starting from a basic graph, then deriving from the *arc diagrams*, we found a meaningful visualisation of OMax knowledge and functioning. This representation may appear a little confused at first sight, however, we will see 2.4 that it is possible to extract more concise and higher level information from it. When displaying this representation to the musician while he is playing, we chose carefully the more adapted visual and explain it.

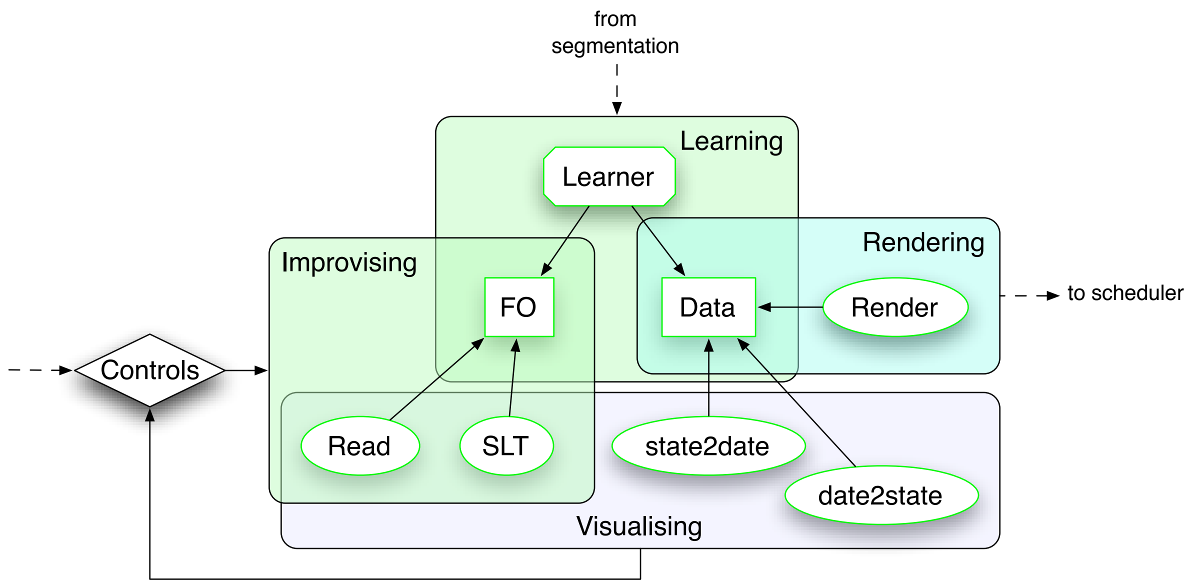


Figure 2.11: Architecture of externals (green framed). This scheme replaces the OpenMusic part in the whole architecture Fig. 1.7

2.2 Design & Development

To search for significant visualisations, the access to all the information contained in FO was necessary. We needed to be able to browse this knowledge with possibly different strategies of navigation to extract relevant data depending of the goal. Adding further to this a long term project of the Musical Representation team for rebuilding the OMax architecture to allow more flexible configurations, we decide to develop the tools needed to integrate FO and the improviser heuristic directly in Max/MSP.

The architecture design is oriented towards modularity to improve flexibility and allow various setups. Our development includes objects for Max/MSP called *externals* and *patches* which are the “programs” in such an graphical programming environment as Max/MSP. Externals are programmed in C++ and use the C Application Programming Interface provided with Max 5.

2.2.1 Architecture

As described in [5] several configurations of agents can be imagined to build an improviser on FO. The structure we designed needed to be as flexible as possible to be able to test several of these configurations. However, the nature of FO is a graph representing the internal organisation of a sequence; once built, the graph itself is (almost) independent from the data it represents. In order to provide general tools to build FO on different kind of data (in our case musical descriptors but also letters), we kept separated the data and the FO build from these (see architecture Fig. 2.11). Furthermore, this principle allows future research to replace FO with another memory model on the same data or use alongside several memory models.

2.2.2 Externals

As shown Figure 2.11, we developed 8 externals for Max/MSP (in fact 9 as we will see later) to implement and use FO in Max/MSP. We can distinguish three types among them: data structures (rectangle boxes), writing objects (bevelled box) and reading objects (round boxes).

There are two different data structure objects: `FO` and `data`. The `FO` structure implements the whole Factor Oracle graph as a vector of states. Each state includes its number, a list of transitions starting from this state, a suffix link and the associated *lrs* and also a list of *reverse suffix links* [3] which are all the suffix links arriving to this state. The reverse suffix links are necessary to efficiently compute SLTs. This external is totally independent from the type of data it represents. The `data` data structure is a "template object" which currently has to forms: `LETTERS` and `MIDI_MONO` (a third form: `SPECTRAL` is being developed) corresponding to the type of descriptors it will embed. This structure holds all the musical data of each state of FO. It also include hash tables to efficiently translate from date references to state numbers and the other way around. Independently from the form, each state includes its state number, the reference in the recording buffer (in milliseconds), the duration (ms), the phrase number (given by the segmentation module), the section number (idem). Naturally these data are not significant for letters FOs but conveniently for test purposes, the buffer reference in a letter FO is its number+1 and its duration is 1. Of course, monophonic MIDI states include three parameters: pitch, velocity, channel as specified by the MIDI protocol.

Writing objects are responsible for storing data into the data structures. The `learner` object is bound to an `FO` object and a `data` object and add the states simultaneously in both of them. It includes the construction algorithm describe in 1.2.1 to build FO. Because it requires a comparison function, this object is also "typed" with one of the previously mentioned forms. The current comparison for pitches is done modulo an octave (12) which creates more patterns repetitions — but we will see later that we still consider octaves in improvisation mechanism.

Another writing object, `build`, which does not appear on Fig. 2.11 is implemented. Bound to a `data` data structure only, it stores states without building the corresponding FO. It allows to memorise improvisations computed by the improviser module and consider them exactly of the same nature as the original sequence. Among other possibilities, we can then replay improvisations or build new FO on these.

Once data are stored in data structures, five objects allow us to retrieve information. `read` object, bound to an `FO` object, is meant to read and output the data of any state in FO. `render` object is the exact equivalent for `data`. `state2date` and `date2state`, bound to a `data` object permit to convert very quickly state numbers into dates (outputting the starting and the ending date) and dates into state numbers.
Regarding the `SLT` object — we extensively use in our system — it has two main functions. Bound to an `FO` objects, it is able to browse the graph following suffix and reverse suffix links. Given a state number and a minimal context length, it computes and output the pruned SLT which includes the input state. It is also able to calculate and output only the path from a given state to the root of the SLT (considering the minimum context length) or the whole path from this state to state 0. Comparing two of this paths corresponding to different states allows to obtain their lowest common ancestor in the SLT and the exact path in the tree (see Fig. 1.3, 1.5 and 2.8).

2.2.3 Patches

Since our environment developed for FOs does not need any LISP code but is included in Max/MSP, we also adapted the existing version of OMax. From a modular version of OMax named MoMax (Modular OMax) and developed recently by Assayag, G. — still using a LISP/OpenMusic core to handle FO — we split the unique patch into functional entities and add our modules. The architecture is graphed Figure 2.12; we named our system WoMax, standing for Without OpenMusic OMax.

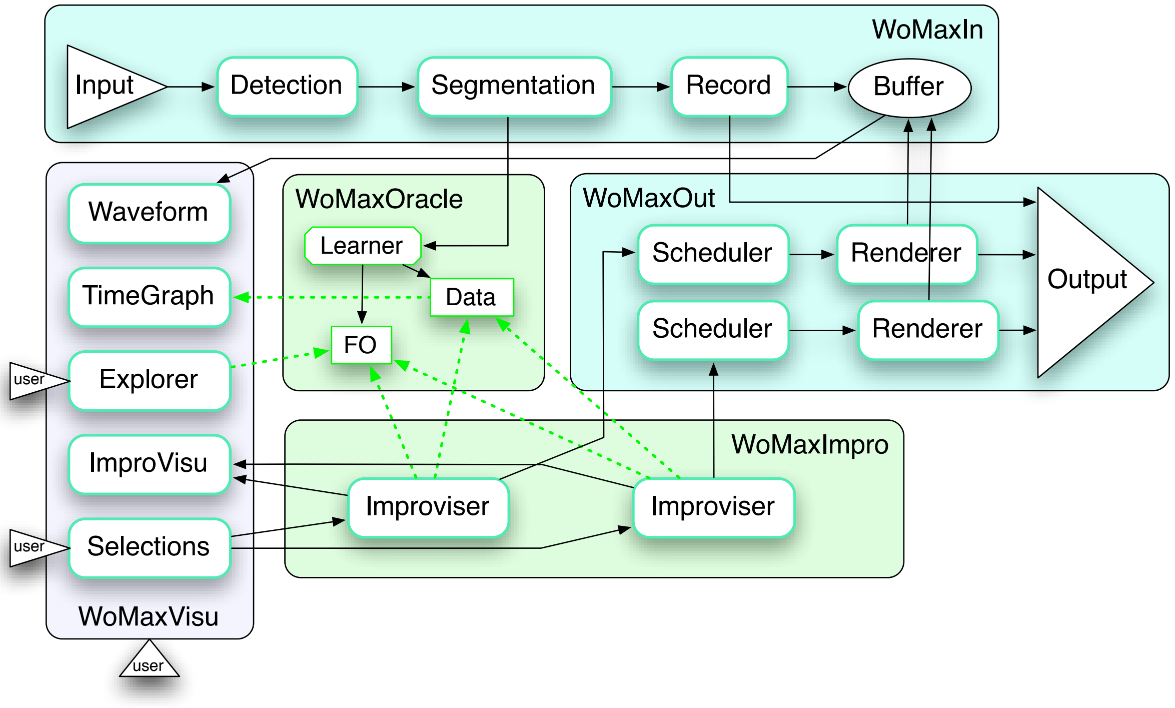


Figure 2.12: Overall architecture of our system: WoMax

Modules with a blue-green background are part of MoMax we used without any changes except splitting or duplicating some patches. In the same way that Max/MSP world exchane data with LISP world in OMax to process the audio stream, these modules handle all the audio part of WoMax.

The very green background indicates FO modules. `WoMaxOracle` is the learning system. It builds the knowledge from segmentation data and sends to every module the current length of FO. Thus every modules is aware of new available data. Reading FO and data structure is done internally thanks to the externals described in 2.2.2. This is shown with green dashed arrows. `WoMaxImpro` is the generation module. `Improvisers` decide when and where to jump by browsing the knowledge (see 2.3 for more details). It receives parameters from the interactive visualisation, in particular, regions of the knowledge to allow or reject. It also sends the current improvisation states and the next jumps to the visualisation (`ImproVisu`). Since all the modules are patches constituted of sub-patches (blue-green-framed), we can easily duplicate some functional parts of the system. Currently we use two `Improvisers` browsing the knowledge independently and able to sound simultaneously (mixing is perform by the WoMaxOut module which is a modified version of MoMax output. See 2.3 for more details on `Improvisers`).

Finally, the visualisation module (with a light blue background) is in charge of the Graphical User Interface based on the visuals explained in the first chapter. The `TimeGraph` sub-patch draw the skeleton with the basic suffix links representation. The `Explorer` allows the user to display arcs, triangles and contexts at mouse position, on clicks, "on-line" (i.e. each time a new state is learnt) or for all the states. The global display is interactive, the user can zoom in/out and move along the graph. We also implemented a system of selections (with the mouse) on the graph. These regions are sent to the `Improvisers` and allow to lock them in a chosen part of the knowledge (see 2.3). This basic control was already included in OMax, however we enhanced it and add new possibilities.

An overview of WoMax interface is presented Figure 2.13.

2.3 New Heuristics

Our modular architecture inside Max/MSP allowed us to redesign the improvisation heuristic. To be able to compare, we first implemented the same heuristic as OMax with a functional grouping to refine it and derive from it later on. Figure 2.14 shows the organisation we adopted.

2.3.1 Strategy & Descriptors

Where the LISP code was at once finding, selecting and ordering the solutions for efficiency purposes, we chose to separate these functions. The `SLT` path first collects all jump candidates. Possibly, to anticipate and avoid to miss a high quality jump, we look for solutions not only from the current state but in a window of several states ahead starting from the current one. Then these solutions are selected. We currently use two criteria: the localisation of the solution if the region-locking is on (`Regions`, see 2.3.2) and the `Taboo` list to avoid loops (explained in [3]).

Then we compute descriptors on accepted solutions to evaluate their quality and order them. We currently consider the precise `Context` length (*cl*) shared between the origin of the jump and the target — as we have seen in the first chapter, this common context can be longer than the pruning threshold of SLT. Like OMax, we calculate a `Rhythm` coefficient given by the formula:

$$rc = \left|\log \frac{d_o}{d_t}\right|$$

where d_o is the duration of the common pattern at the origin of the jump and d_t the duration of the pattern at target point. Better rhythm coefficients are smaller ones.

We tried new descriptors, for instance we implemented a simple `Octave` flag (*o*) which equals 1 when the jump candidate correspond to the exact same note (same octave) as the note we jump from, 0 otherwise. We also tested a value indicating the number of states the improviser has to read before jumping. Indeed, as the algorithm usually looks for jumps in a window of several

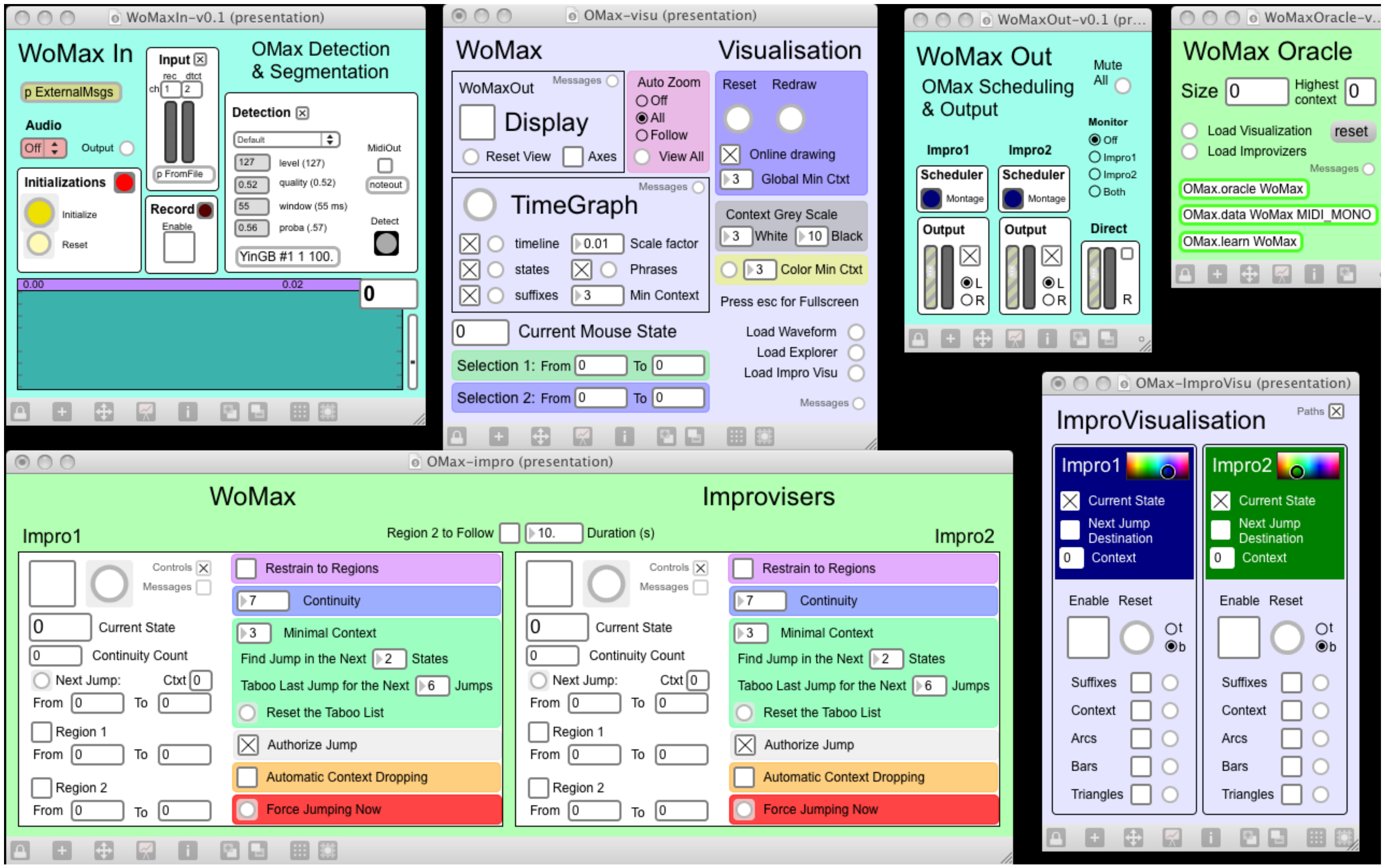


Figure 2.13: WoMax interface without Jitter display window

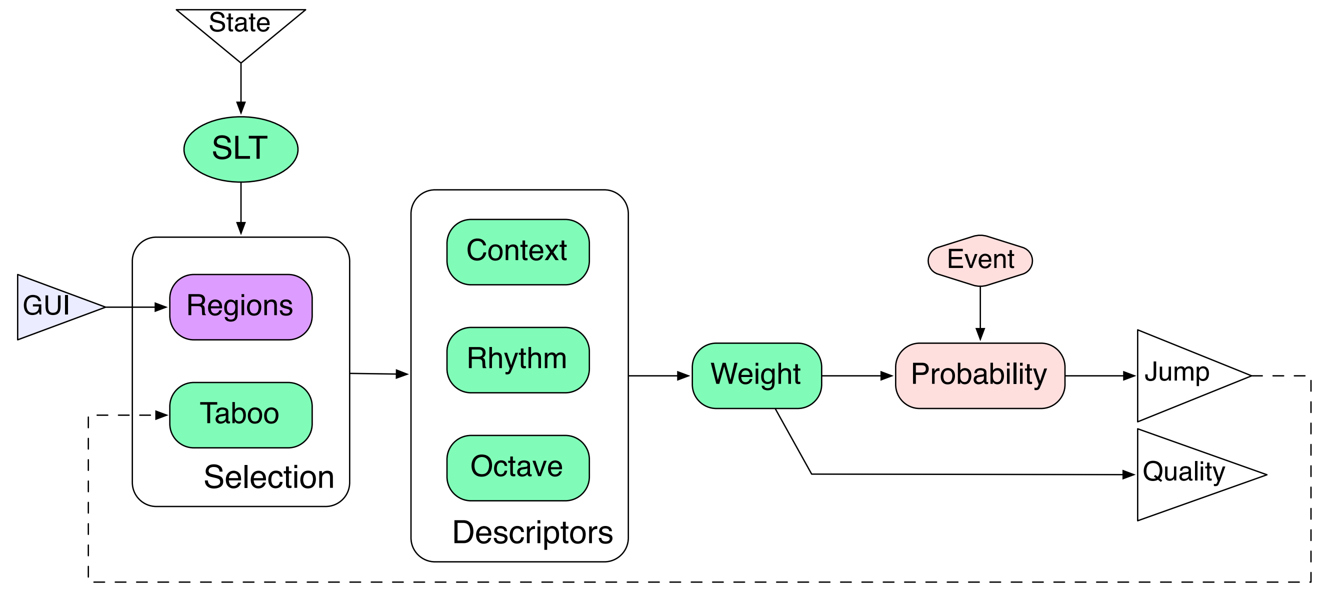


Figure 2.14: Architecture of WoMax `improviser`

states ahead from the improvisers current position, a jump from a state x to a state y with a context m will produce the same notes as the jump from $x + 1$ to $y + 1$ with the context $m + 1$ if it exists. However, it is not musically equivalent because it changes the number of contiguous states to read. A longer continuity gives a better sound result.

Once all these descriptors are gathered for every solution, we combine these in a weight to obtain an overall quality of every jump and order the candidates according to this criterion. Currently the weighting formula is :

$$w = \max(cl + o + (1 - rc), 0.0001)$$

we floor this weight to avoid improbable candidates. Then a jump is picked (among the first 100 solutions maximum) following an exponential probability curve encouraging the best quality candidates.

This system of weight is very flexible and allows to invent and test very easily new strategies of improvisation. We can for example give more weight to the rhythm coefficient which will be musically interesting in a pulsed improvisation. We can also invert and raise the weight of the octave flag to force the virtual improviser to jump from an octave to another etc. Compared to the LISP/OpenMusic improvisation heuristics, this new flexibility deeply changes the role of visualisation, interface and interactive control of the virtual improviser.

2.3.2 Interaction

The parameters of WoMax improviser can be controlled and applied immediately because it computes the next state (consecutive or jump) at once when triggered — LISP improviser of OMax is actually planning the states ahead around every half of a second (without any possibility of going back). Therefore we can influence quickly and efficiently WoMax improvising strategy and orient

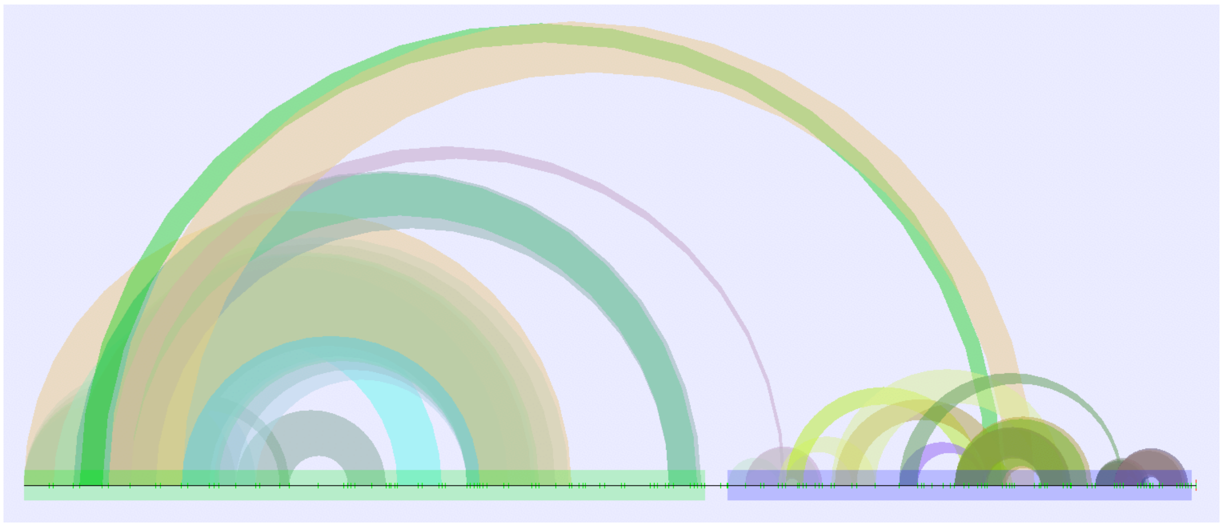


Figure 2.15: Two selected regions (in green and blue along the FO skeleton).

the musical result. For instance a very small *continuity* (i.e. the number of consecutive states to read before trying to jump, parameter already available in OMax) will produce a very dislocated improvisation while a higher value will give the impression of a very continuous musical speech. WoMax enhances these possibilities with a way to force jumping directly and immediately right from where the improviser is and a mechanism of automatic minimal context dropping to accelerate the finding of a jump in a region with poor (in terms of *lrs*) links. In this context, visual feedback on the knowledge is a very powerful tool.

On the current visualisation, we provided the possibility of selecting two regions in the graph (Figure 2.15). These selections are sent to the improvisers where a choice is given to force the jumps inside one, the other or both regions. This mechanism existing with one region in OMax and developed to two regions in WoMax (and may be applied to any number of selection in FO) allows to use more deeply the precise nature of navigation in FO to hybrid two chosen parts of the original sequence by jumping freely from one to the other. This gave very good musical results.

2.4 Clustering

2.4.1 Visual Observation

To go further in the use of selections, it is crucial to identify different parts in the knowledge to be able to chose relevant regions. Now, we can notice on Figure 2.15 which shows 60 seconds of a saxophone with visualisation of all the SLTs pruned with a minimal context of 4, that the two selections correspond to an actual grouping of arches. Considering what arches indeed mean, this grouping represents a higher level analysis of the music: a first theme is played then repeated shortly after which creates thick arches in the green selection (left half). Then a few new patterns are encountered. They are transformed and recombined locally and with two elements of the first

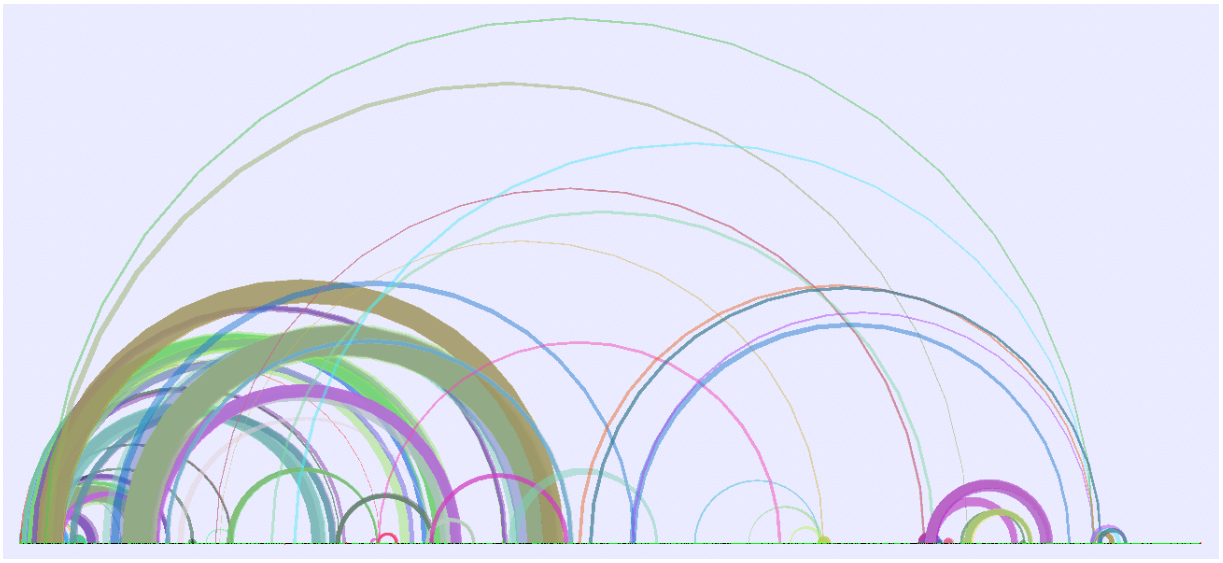


Figure 2.16: Saxophone improvisation of 6’25” with SLTs with a minimal context of 6

theme. This creates the group of small arches in the blue selection with two wider arches starting from the left half.

On a larger scale, this organisation in the arches (naturally likewise in FO) is even more obvious and significant. Figure 2.16 gives an example of 6 minutes 25 seconds of another saxophone improvisation on the standard jazz theme *Bahia*. We clearly identify three very different regions. The first region encountered (left half) is constituted of many thick arches which denote numerous thematic elements and repetitions. The second region (3d quarter starting from left) is almost empty of arches and corresponds to a more free improvisation in very wide range of notes. Finally a third region can be observed (last quarter) with more local and thick arches. This last region is actually constituted of to parts (which we can also distinguish), both very repetitive. The first group of small arches is link to a very abstract moment with rather long and repeated note with spectral effects (multi-phonics) then a the second group of (even smaller) arches is a small number of very short patterns repeated numerous times as a *groove* or an *ostinato*.

2.4.2 Extraction Algorithm

From this observation showing that very consistent and high level information is contained in FO, we search for an extraction algorithm. The extraction of such information can be summarised as a problem of graph partitioning or clustering. However one of the main interest of FO is its incremental construction therefore we wanted to keep this advantage even for higher level information matter. This excludes usual clustering methods using similarity matrices, eigenvalues or expectation-maximisation algorithms as those presented in [7].

In the FO construction algorithm we can observe a very strong reinforcement mechanism since links with high *lrs* (long patterns) can appear long after the first occurrence of the pattern and make regions emerge at once. An extraction algorithm needs to take advantage of this mechanism.

Furthermore another very notable observation is the difference between "small" or *local* arches that is links pointing to recently appeared patterns and wide or *global* arches which represent the replaying of theme presented long before. Both types of links, if they appear closely permits to define consistent regions.

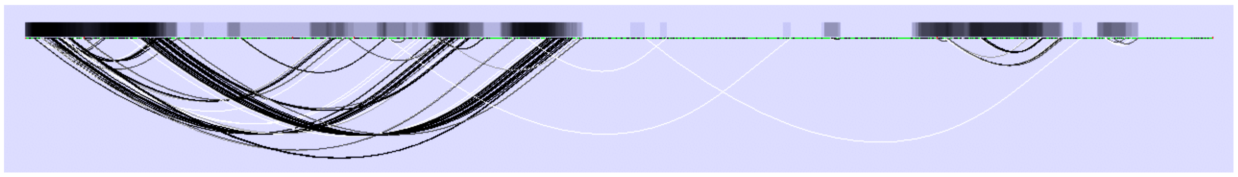
The proposed algorithm is based on a weighting system which allows the reinforcement mechanism and gives, with a threshold, boundaries to regions. It uses a list of weights (integer) associated with every state (and initialised to 0). It is linear in time and space. Three control parameters are required:

- *minCtxt*: a minimal context length. Only links with a rather high *lrs* are to be considered (usually around 6).
- *local*: a value to decide the local/global character of a link. This value is a criterion on the span of the link (number of state between both ends). It is used as a coefficient on *lrs* and is superior to 1 (a value inferior to 1 would declare as local only a pattern overlapped with its own repetition, that is a loop. Usual values are from 2 to 15).
- *w*: a value to define a zone of "influence" of a link. This value is also a coefficient applied on *lrs* and superior to 1 since the whole pattern must be included in the influence region (usually between 1.5 and 2).

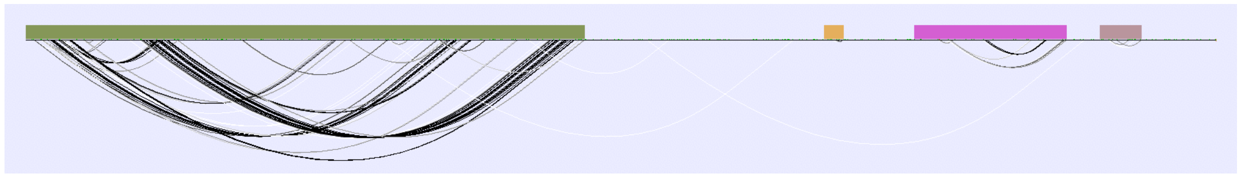
Defining $W(i)$ the weight, and assuming $W(k)$ is known for all k from 0 to $i-1$,
defining I a temporary interval or union of interval of states,
we consider state i, its suffix $S(i)$ and context length $lrs(i)$:

```
If the context of the link is high enough:  $lrs(i) \geqslant minCtxt$ do:
If the link is considered as local:  $i - S(i) \leqslant local$
    Then there is only 1 influence zone containing both ends of the link:
        $I = [S(i) - w * lrs(i), i]$
    Else the link is considered as global:  $i - S(i) > local$
        there are 2 influence zones containing each end of the link:
        $I = [S(i) - w * lrs(i), S(i)] \cup [i - w * lrs(i), i]$
For every state in the influence zone:  $\forall k \in I$
    Increase weight:  $W(k) = W(k) + 1$
```

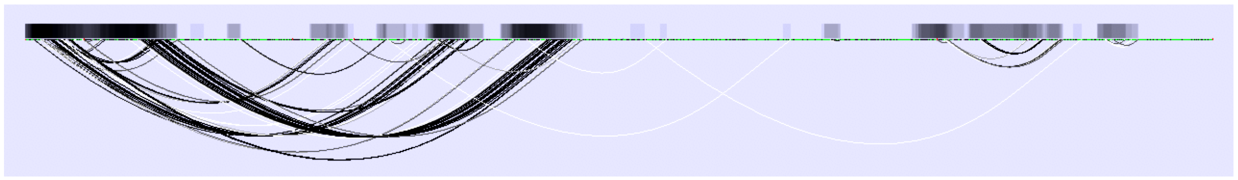
A visualisation on a grey scale (rectangles above the timeline) of the weighting algorithm is given Figure 2.17a and 2.17c. Regions and their boundaries are given by contiguous states with a weight superior or equal to 1. However, to keep only strongly consistent regions we use a threshold (W_{min}). Be aware that to avoid cutting in the middle of a musical pattern, we still calculate the boundaries of selected regions including the states of weight non equal to zero but inferior to the threshold. That is we calculate boundaries with all weights then we keep only regions with a maximum weight superior to the threshold. This is shown Figure 2.17b and 2.17d. Changing the *local* parameters changes the number of regions. With high values of *local* (Fig. 2.17b) a rough clustering is done and we find the parts with observed previously with the arc representation. Lower values of *local* parameter (Fig. 2.17d) reveals a refined analysis where we can read in the first half, three important recalls of the first theme after a brief development.



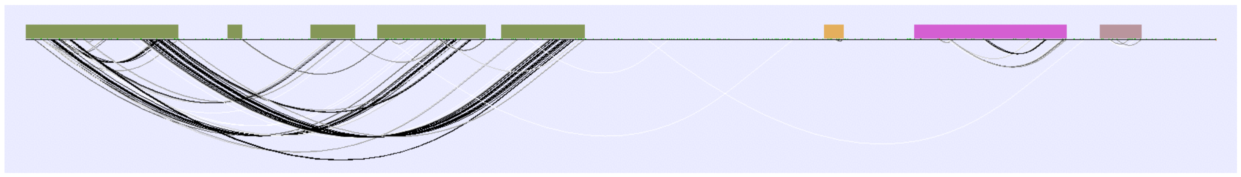
(a) Visualisation of the weighting with *local* = 21



(b) Visualisation of the clustering with *local* = 21



(c) Visualisation of the weighting with *local* = 12



(d) Visualisation of the clustering with *local* = 12

Figure 2.17: Visualisations of clustering done on the same 6'25" saxophone example with $minCtxt = 6$, $w = 2$, $W_{min} = 3$ and 2 values of *local* (suffix links with $lrs \geqslant 6$ are graphed in the lower half)

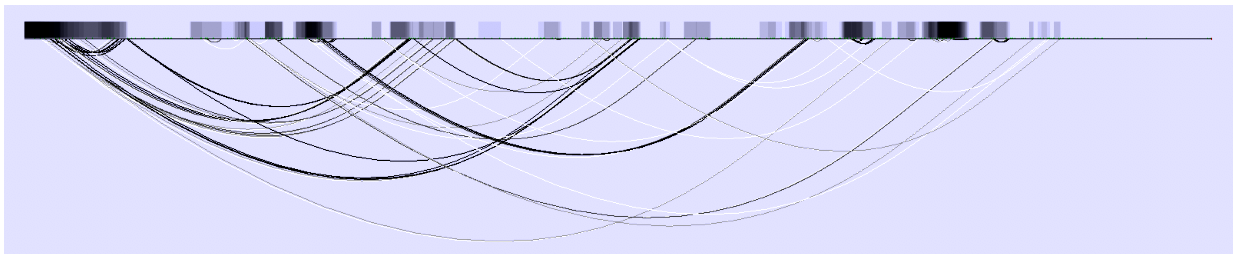
Considering the nature of a suffix link, it is rather easy to add to our algorithm a rough identification of regions and be able to link a region with another when they have the same thematic content. Nevertheless, some situation are, even on a musicology point of view, difficult to determine. For instance, consider a piece with a first theme A and later on a second theme B at first clearly separated from A. Now in the development of the piece, A and B are chained one to the other with a subtle musical overlap as we can find in many pieces. How to define a precise border between the second occurrence of A and the second occurrence of B ? Even though we are able to identify A and B both times, we can not draw a precise border and two different musicology analysis will probably place it differently. Our identification encounter the same problem. To decide nevertheless and simplify the algorithm we chose to take the more consistent region, that is the region with the highest weight. Here is the modified algorithm:

Using the same conventions as before,
defining $r(i)$ the region ID and assuming it is known for all states with a non zero weight,
using a global region counter r_g (initialised to 0), a local region ID r_I and a local list of region ID lr_d,
we consider state i, its suffix $S(i)$ and context length $lrs(i)$:

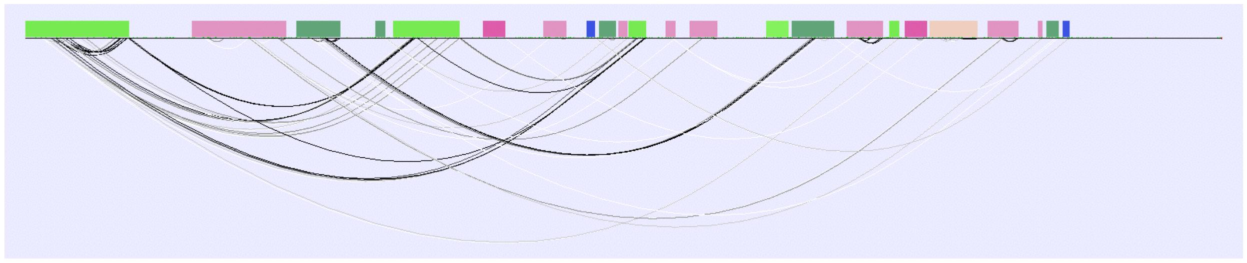
```
If the context of the link is high enough:  $lrs(i) \geqslant minCtxt$ do:
If the link is considered as local:  $i - S(i) \leqslant local$
    Then there is only 1 influence zone containing both ends of the link:
        $I = [S(i) - w * lrs(i), i]$
    Else the links is considered as global:  $i - S(i) > local$
        there are 2 influence zones containing each end of the link:
        $I = [S(i) - w * lrs(i), S(i)] \cup [i - w * lrs(i), i]$
Determine the region ID: $r_I = FindID(I)$
Store the list of ID to drop:  $lr_d = DropID(I, r_I)$
For every state in the influence zone:  $\forall k \in I$
    Increase weight:  $W(k) = W(k) + 1$
    and Assign the ID: $r(k) = r_I$
For every state of FO which has a non zero weight except the influence zone
(already done):  $\forall k \in [0, i] \backslash I \quad / W(k) > 0$
    If the region ID is in the drop list:  $r(k) \in lr_d$
        Then Overwrite ID: $r(k) = r_I$

FindID($I$) =
    Find the state with maximal weight in the influence zone $I$:
        $i_{max} = argmax_I(W(i))$
    If a non zero weight is found:  $W(i_{max}) > 0$
        Then Return its ID: $r(i_{max})$
    Else no state in the influence zone is already in a region:  $W(i_{max}) = 0$
        Return the current region counter:  $r_g$
        and Increment it:  $r_g = r_g + 1$

DropID($I$, $r_I$) =
    List all the region ID present in $I$ (if any):  $lr_d = W(I)$
    Remove doubles in the list:  $unique(lr_d)$
    Remove the chosen region ID (if present):  $remove(r_I, lr_d)$
    Return the list of ID to drop:  $lr_d$
```

A visualisation of this modified algorithm is presented Figure 2.17b, 2.17d and 2.18b. Different colours represent the different regions recognised. The identification mechanism associating similar regions, allows to obtain a new definition of region. A “region” does not need to be contiguous any more and can gather several parts spread along the sequence as shown Figure 2.18b. With this system the clustering goes beyond the time proximity and local consistency to recognise the actual musical content.

Further than just navigation in FO, combining patterns to generate new melodies, our first observation and extraction of information on a higher level organisation shows that this compact structure can be used for many more purposes. FO incrementally constructs very consistent knowledge which needs to be studied from different level and point of views. Our algorithm is a first step in this direction.



(a) Visualisation of the weighting



(b) Visualisation of the clustering

Figure 2.18: Visualisations of clustering done on another saxophone example (3'38") with $minCtxt = 6$, $w = 1.5$, $W_{min} = 3$ and $local = 6$ (suffix links with $lrs \geqslant 6$ are graphed in the lower half)

Designing and implementing tools to explore FOs led us to reconsider our view on OMax and the use we make of its internal structure. We found some meaningful representations of the FO graph and investigated ways to benefit from the new accessible matter. The renewed access to such a knowledge enhanced the control possibilities of its use in the improvisation context and created a need of flexibility in the functioning of OMax. With WoMax, we provided such a new flexible experimentation environment, consistent with the visual feedback. But this new view on FO also opened perspectives on different approaches to this pattern representation. We abstracted a very new clustering algorithm which revealed a significant potentiality and a very musical relevance.
After a more distant evaluation of our results, we will examine the future directions possible for our work.

Chapter 3

Results & Evaluation

3.1 WoMax

Our research on visualisation led us to reconsider and rebuild a new experimentation prototype. This development produced a flexible environment which modular structure allows us to try many configurations. It does not use any more LISP code and the OpenMusic component. Yet it is compatible with the previous systems OMax and MoMax. WoMax sole does not use either the OSC protocol anymore, every messages being sent inside Max/MSP. But with the modular structure, we can conveniently reintroduce this protocol to distribute tasks over several computers.

The improviser we currently implemented works essentially as the OpenMusic/LISP improviser. But some refinements have been included and we structured this function according to the modular character of Max/MSP. Therefore the testing of new heuristic became easy. The deepest and very significant improvement in this new improviser is its responsiveness. It now calculates at once, once triggered, the new location to play and does not necessarily anticipate more than one note ahead. This allows the sound man controlling the computer based improvisation to react almost as quickly as an acoustic musician which renews dramatically interaction possibilities.

We also developed our system with two of these new improvisers passing from a duo to a trio with the musician and two virtual improvisers. Yet it is most of the time more interesting to alternate between the two improvisers differently configured to quickly change the musical content (see 3.4). The visual feedback which was our starting point took us also to expand the controls over the improviser. As many controls as new refinements have been included creating a real interface panel (Figure 2.13) to finely tune the improvisation.

3.2 Visualisation

From a simple timeline which shows only the states of FO to a very complex display with region clustering and arcs, our visualisation allows to access to the content of FO.

3.2.1 On-Line

A linear time representation from left to right grounds our visualisation on a usual music visualisation paradigm and make it usable for on-line display. As we have seen, the basic link representation, even though it is very elementary is useful to spot interesting regions of the sequence. And we can explore thoroughly these regions with the different colour representations: triangles, arcs, regions.

When working on-line, it is very useful to visualise at once (with arcs) the incoming patterns and their other occurrences in the past so we can come back to these later on by selecting the right region. The clustering and its ever evolving regions give also significant data on the organisation of the sequence. Its reinforcement mechanism shows the very musically collapsing of thematic elements into one consistent region or the spread of some thematic elements over the whole past.

Naturally, the gathering of all the visual elements creates an overly complex visualisation. But we are able to chose precisely what we would like to show therefore adapt our display to our need. When showing it to the musician for instance, we may keep only arches on top of the timeline which are visually easier to understand or display the region clustering.

3.2.2 A Posteriori

As we can load files into OMax, MoMax and WoMax instead of the real-time audio input, we may use the visualisation as an off-line exploring tool. The visualisation can be zoomed in and out and the user is able to scroll along the time axis which makes easy to use this tool to discover a piece. With FO and its representation we can search for a particular pattern, examine the context of its repetition, play and look to its different durations etc.

In an off-line analysis, the clustering algorithm may reveal several level of analysis by testing different parameters. Extracting by turns very high level structure and spreading of very small thematic elements we may draw a very consistent organisation of the file.

3.3 Musicians

During this work, we had the opportunity to test our visualisation system in session with musicians. The main system used during these session was OMax but we tested on another computer our system and we shown and explained to the musicians our visualisation.

3.3.1 Fabrizio & Aka Moon

The saxophonist Fabrizio Cassol and his experimental jazz band Aka Moon gave a concert on the 8th of June (Agora Festival, Centre Pompidou) integrating OMax (on the saxophone only) in their very structured compositions. Therefore we could attend and try our visualisation during two sessions; one with the saxophonist alone, another with the whole band. During the first session, our visualisation appeared to be a very good support to explain the functioning of OMax. Indeed we can show step by step how OMax builds its knowledge and how its uses it to improvise. It was

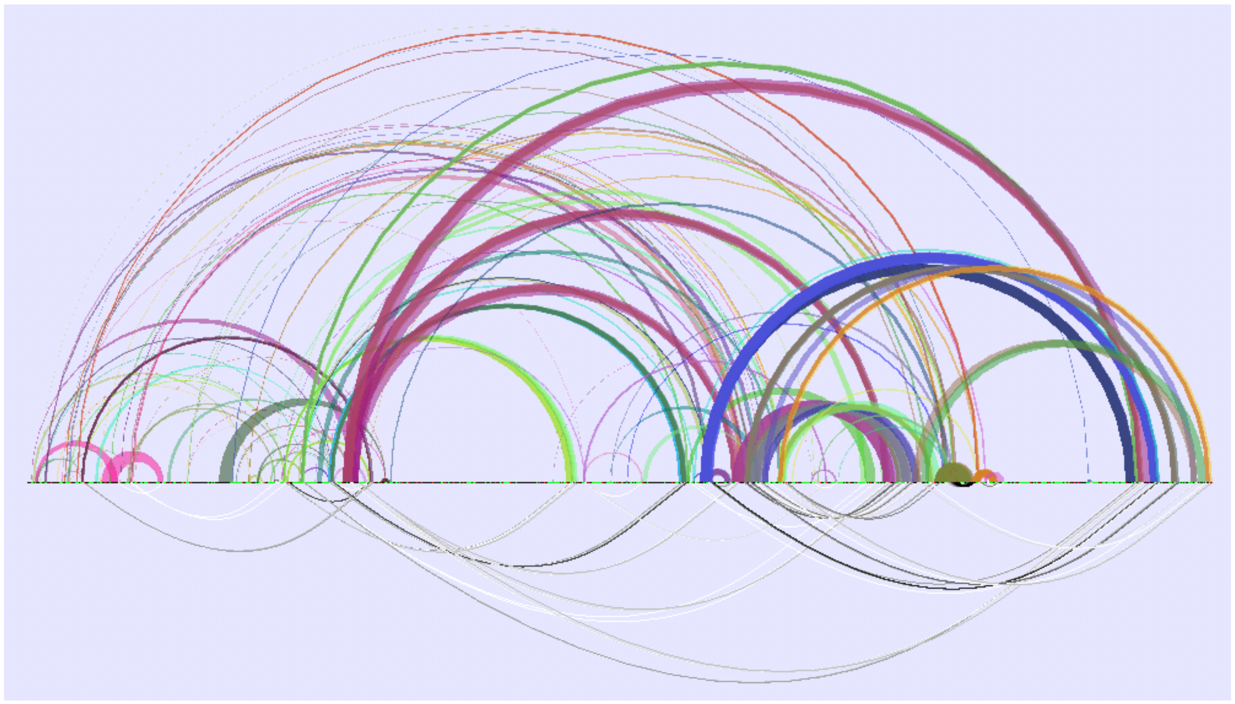


Figure 3.1: 11’37” of the session with the jazz band Aka Moon. FO is built on the sole saxophone and the minimal context displayed is 7

very helpful to make the musician understand. During the second session, the main difficulty was to combine OMax with the band. We could not show in real time our visualisation because of the music sheet and preparation for the concert (without visualisation). But we recorded the session and Figure 3.1 is a visualisation of 11’37” of it. Here again, the intuitive idea of our clustering based upon a notion of density of high *lrs* links appears clearly, so does the global/local character of links.

3.3.2 Cécile & Vincent

Cécile Daroux, flutist and Vincent Courtois, cellist came the 30th of April specifically to try the OMax system. One improviser was put on each of them and they tried several configuration: one instrument and the improviser, two instruments but only one improviser and the quartet of both musician and both improvisers. Our visualisation was put successively on each of the musicians. After a first test without any knowledge on the OMax system, we explained (thanks to the visualisation) how it works and displayed along the session on-line visualisations or hindsight results. Figure 3.2 shows a 3’20” improvisation of the flutist with OMax. The visualisation was not shown to Cecile during her improvisation but she built naturally a very symmetrical structure as we could notice afterwards (Fig. 3.2).

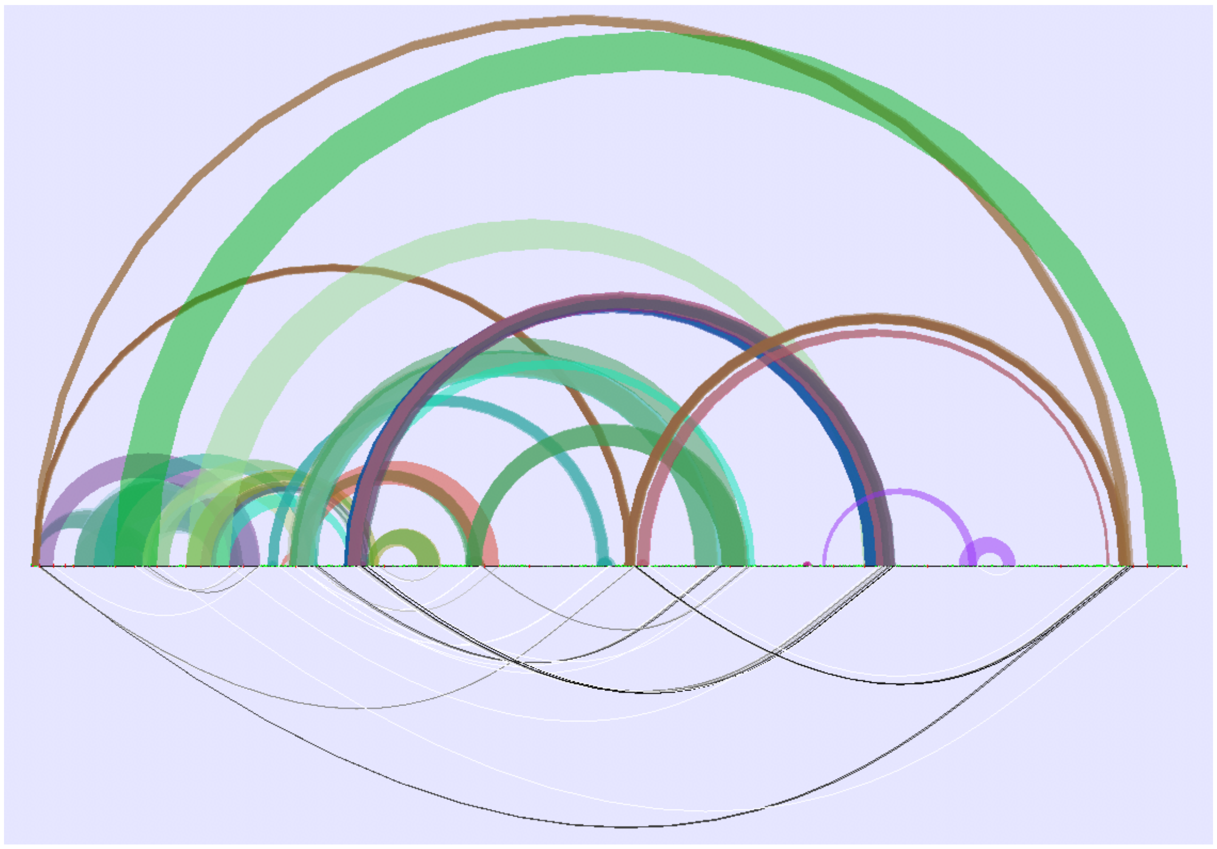


Figure 3.2: A 3’20” improvisation for flute and OMax

Vincent, the cellist, tried the OMax system after Cécile and didn’t know at all what to expect and how the system works. After a single try with his virtual improviser only, they improvised together with both musicians and both virtual improvisations. The very wide range of possibilities of the cello is very remarkable with our visualisation. A very melodic improvisation gives large, spread arches as shown Figure 3.3 while a very abstract improvisation with precise moments where the cello take the role of a rhythmic support with *ostinati* (very repetitive and pulsed patterns) is obviously readable on Figure 3.4.

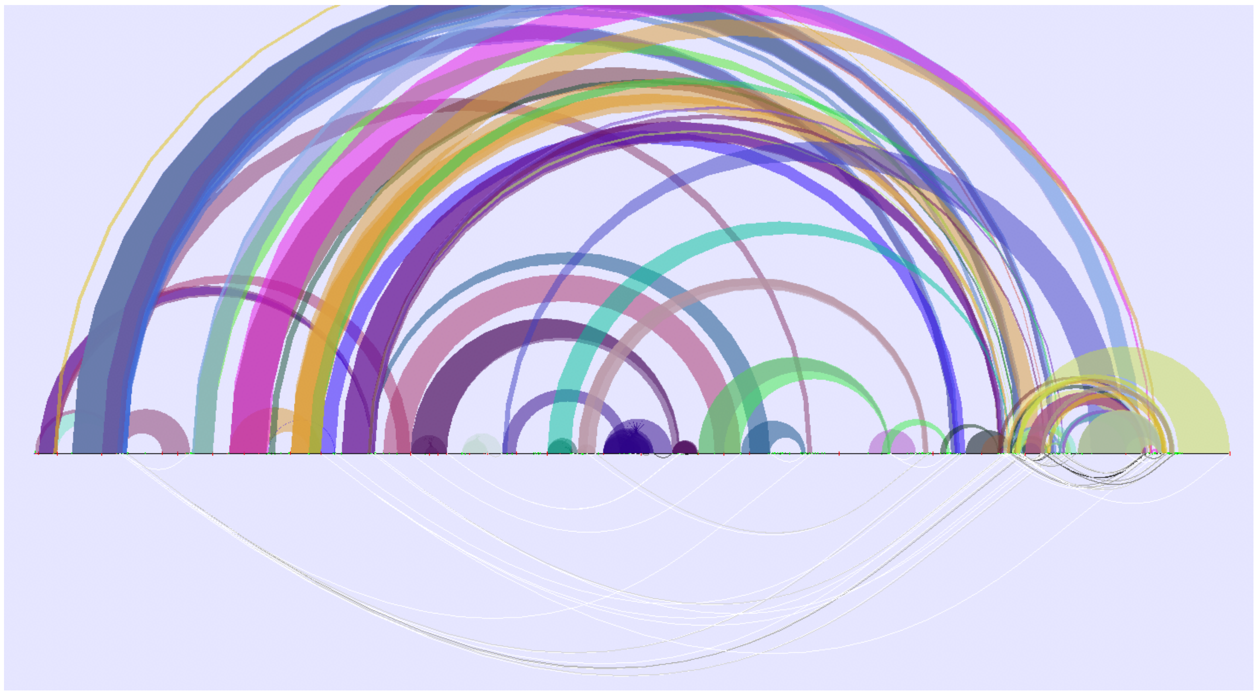


Figure 3.3: A 3’16” melodic improvisation for cello and OMax

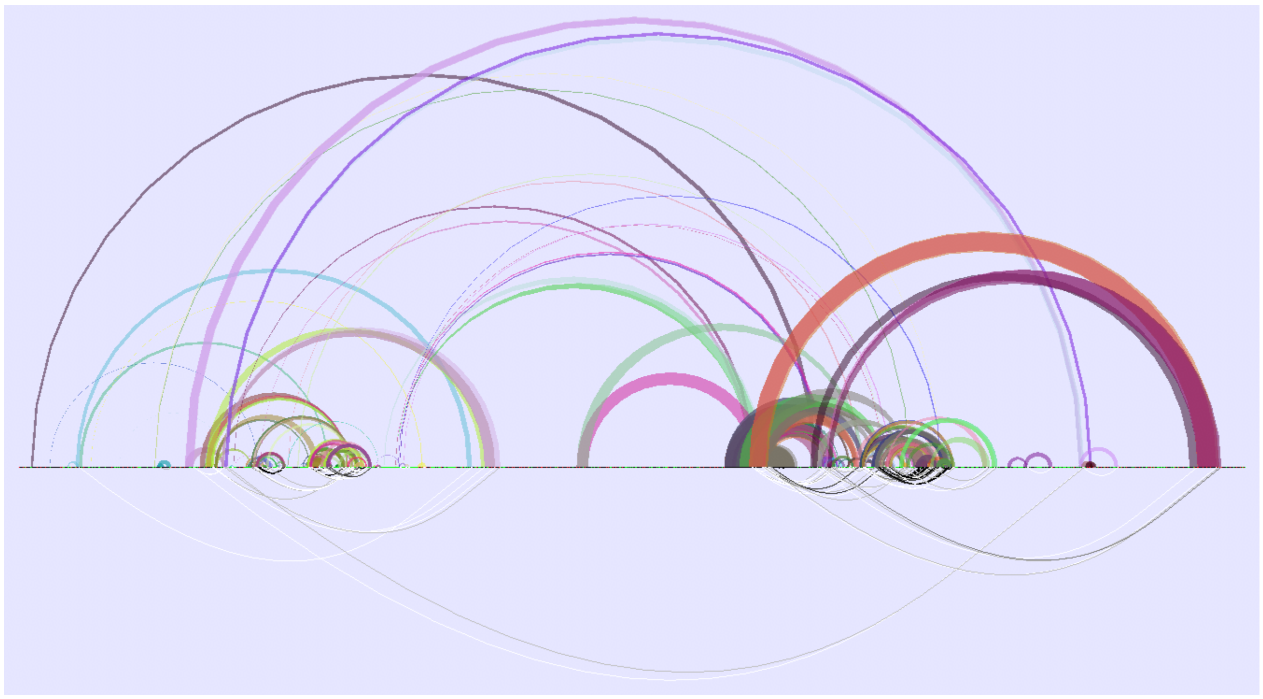


Figure 3.4: A 12’02” quartet improvisation for flute, cello and 2 virtual improvisers

3.4 Personal Testing

3.4.1 As a cellist

During this internship, I was very curious about the feeling such a system could bring to a musician. It took me two month to finally try MoMax with my cello (but without any visualisation except the waveform — I play the cello for 19 years). Working on OMax and FO every day I actually could not test it ingenuously abstracting my knowledge of its functioning. But that gave me all the more delight to play with it. I configured a pedal board to be able to control a few parameters as the region, the continuity etc. And played a duet with my self. I realised shortly that I naturally knew what input would output a interesting result and always comfort my improvisations in this direction. Unfortunately I had no time enough to explore further the musical possibilities of the system.

But playing another time with the same setup with other musicians (a pianist, a guitarist), it expanded very much the range of my musical discourse. I could overlay two ideas, explore one of them with MoMax and develop the other one with my cello, mixing them afterwards by nature of this system. That was actually very gratifying. I guess, further exploration will reveal more subtle use of the system and difficulties to have a longer term construction. I will for sure keep on trying it !

3.4.2 As a WoMax Player

From June WoMax was stable enough to be tested in a real improvisation as we did with OMax. I tried it on the 2nd of June on a clarinet playing in a improvisation workshop with a tenor saxophone and piano. I was, this time, only controlling WoMax with its visualisation on my computer. The clarinetist had no control at all and no visual feedback was provided to any of the musicians.

I tested the new improviser and its reactivity and the double improvisers system (two improvisers in parallel on the same FO) both with the visualisation on my screen. The reactivity is a really important improvement. Compared to OMax or MoMax it provides a true control over the musical dialogue and makes WoMax more a new instrument reusing the sound of another than a shadow from one of the musician. It enables the person controlling WoMax to react almost as quickly as the other musicians, which is, in the improvisation context a very important quality.

The visualisation — and the habit of reading it — is also a real help to recall intentionally particular passages and once again orient the musical speech as a musician would do. The double selection is very useful because it allows to chose on purpose the musical content to recombine thanks to FO. That gives very pertinent results and allows to introduce new musical ideas the musician had not thought of.

This direction is for me the most promising.

Chapter 4

Future Directions

Working on OMax with a totally new paradigm (visualisation), our research and development seems to open up very diverse and promising directions. From the pure software improving to new information retrieval applications, we will examine some of the possible continuations.

4.1 WoMax

4.1.1 Development

After reimplementing the heuristic of OMax and WoMax in WoMax, it appeared that our modular improviser is slightly heavier than the LISP code which was very optimised. Thus it would be profitable to improve our algorithm and enable the use of WoMax in concerts and production contexts.

Currently, we implemented only letters and monophonic pitch classes in our FO architecture. Including spectral descriptors is very important to improvise in many more contexts: with the piano, with wind players who speak or sing aside or inside their instrument, with groups of instruments or even an orchestra etc. This development is already planned in our FO architecture and should be done shortly.

Another direction is to test the possibility of running several copies of WoMax in parallel. This running already exist in OMax and MoMax and is often used to handle two or three instruments and alternate the improvisation among them. This feature is also included in WoMax but not tested yet.

4.1.2 Visualisation & Interfacing

Now we have invented a visualisation that shows and allows to explore all the content of FO, we are conscious that we need to introduce “intelligence” in our display to improve the immediate understanding. A first step in this direction is to find a logic in the attribution of colours. These colours are currently picked at random but we may start to combine the clustering algorithm with the SLTs display to obtain a more consistent representation.

Considering our work as a first step in visualising OMax, we will also need to adapt these representations to different contexts. WoMax (as OMax or MoMax) can be used by a musician alone who controls him or her self the computer base “double”, in which case it is very important to have a straight forward visualisation with powerful key elements. On the other hand, when using WoMax controlled by a sound man, he can make use of more advanced representations and choices to drive the computer base improvisation with musical choices. Both of these situations require a visualisation and an interface adapted to their respective use. It probably involves combining two or three of our visual elements into one consistent display and collect feedback from different kind of users to improve it.

4.1.3 Improvisation Strategies

One of the most important changes to make to the current improvisation strategy is to consider the beat (if any in the original sequence). With the architecture of WoMax, we will first try to give simply more weight to the rhythm coefficient. Another option is to invent a new descriptor based on rhythm or beat to order the solutions. This may improve the improvisation in a pulsed context. However, to really synchronise the improvisation with the current beat we will probably need to deeply change the improvisation strategy. This is a all new research direction.

Nevertheless, new strategies which make use of several FOs in parallel are now possible in our FO structure. We will try for instance to build at once a pitch based FO and an interval based FO to compare the improvisation results and certainly combine both FOs to obtain a better quality improvisation. Pitch and interval being very close descriptors, it is rather easy to design a system to jump from one FO to the other, however in the case of very different descriptors as spectral descriptors and pitch it is not trivial to combine FOs. A PhD student is currently working on this kind of issues.

Another noteworthy improvement to the improvisation strategy would be the including in the computer based improvisation a direct feedback on what it played and how it sounded in the short term musical context. Currently, the improviser “forgets” what it just played and continues combining states taking only into account new material provided by the musician. Building an FO on its own output could reveal very interesting feedback our improvisation strategy and is a first step toward a “self-listening” computer.

4.1.4 Anticipation

As we reduced the inner foresight of the improvisation algorithm to a controllable number of states, we now have the possibility to forecast any number of step further in the computer based improvisation. Nonetheless, given the probabilistic nature of the algorithm, going back on this forecast is a destructive action; another forecast with the same parameters may output a different path. Thus we will have to define precisely the use of such a forecast, and how we consider its output: is it certain, so we can not go back but we can count on it to anticipate or is it a possibility we can erase to pick another one with the risk of never finding a better one ?

Considering a forecast, we may consider also to foresee the next steps. Choices in the visualisation of these are non trivial and may considerably increase the complexity of the representation.

However, this opportunity opens very interesting interaction possibilities with the musician since he or she will be able to anticipate the computers playing. This would deeply change the relation between the real and the virtual improviser.

4.2 Higher Level Informations

As we mentioned, visualising the FO structure and having an overall view of it showed some higher organisation scheme which correspond to musical structure. We started with the more obvious of them, extracting regions based on the density of long context links.

4.2.1 Regions

Our extraction algorithm takes three control parameters. A first improvement is to automatically adjust these parameters depending on a musical criterion for instance the maximal size of a region or the maximal number of clusters belonging to the same region. For musical purposes, we also need to refine our region identification algorithm to be able for example to add artificial boundaries if required by the musical context (i.e. being able to distinguish between the repetition of themes A and B even if they overlap). And then compare our automatic analysis to musicology analysis.

The input parameters of the algorithm are also efficient to obtain different level of region analysis. Thus we may want to find presets and define precisely these levels. We could also have several levels of analysis at once with different copies of the algorithm and build a more complex but more significant analysis.

A third direction to benefit further of the clustering is to define different kind of regions and being able to distinguish between them. For example, contiguous parts with no high context links may reveal regions where a musical development introducing new musical material is encountered. We need to examine these "anti-regions" to know if they are of any musical significance. Regions with numerous links pointing to a very large number of locations spread over the sequence is another interesting type of region which would be helpful to the improvisation algorithm. They constitute what we could call "hot spots": they provide good starting points to jump anywhere else in the sequence.

4.2.2 Music Information Retrieval

From our visualisation, we abstracted a clustering algorithm which works incrementally. Other works on clustering are mostly done with non-incremental algorithm in the Music Information Retrieval (MIR) community (and others). To evaluate our algorithm we should compare it with some of these off-line methods and check with musicology analysis to have an objective idea of our results.

It would be a pity to lose the incremental character inherent to FO, however it may lead to very powerful improvements. We may combine the idea of our extraction with other clustering methods to get some very new directions in MIR.

Conclusion

Starting from the current version of the improvising system OMax, we studied its core, its knowledge based on a graph: Factor Oracle. FO is a powerful and condensed representation of all the repeated patterns in an input sequence — in OMax a sequence of musical descriptors extracted from an audio stream. Looking for a suitable representation for this knowledge, we found an interesting pattern visualisation: *arc diagrams*.

On this background, we made OMax internal structure visible and invented tools to explore such a knowledge. The necessity to freely browse FO embedded in OMax led us to redesign its architecture and make it modular and flexible. While rebuilding its inner structure, we took this opportunity to reconsider the navigation heuristic in FO to make it also modular and flexible but more responsive as well. We obtained a new consistent improvising environment: WoMax which we started to test. Our new ability to have an overview on FO showed us high level information lying in it. Thus we proposed a new clustering algorithm allowing to extract musically consistent regions based on the links constituting FO.

The three of our contributions: the new environment WoMax, the visualisation of FO and the new clustering algorithm open numerous possibilities in very different directions such as new experimentations with several FOs in parallel, new interfaces which would show an anticipation of the machines playing or new ways to guide the improviser through particular regions spotted in its knowledge.

Bibliography

[1] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot, *Factor oracle: a new structure for pattern matching*, SOFSEM'99 (Milovy, Czech Republic), LNCS, vol. 1725, Springer-Verlag, November 1999, pp. 291–306.

[2] ______, *Factor oracle, suffix oracle*, Tech. report, 1999.

[3] Gérard Assayag and Georges Bloch, *Navigating the oracle: a heuristic approach*, International Computer Music Conference '07 (Copenhagen, Denmark), Août 2007, pp. 405–412.

[4] Gérard Assayag, Georges Bloch, and Marc Chemillier, *Omax-ofon*, Sound and Music Computing (SMC) 2006 (Marseille, France), May 2006.

[5] Gérard Assayag, Georges Bloch, Marc Chemillier, Arshia Cont, and Shlomo Dubnov, *Omax brothers : a dynamic topology of agents for improvization learning*, Workshop on Audio and Music Computing for Multimedia, ACM Multimedia 2006 (Santa Barbara, USA), Octobre 2006.

[6] Gérard Assayag and Shlomo Dubnov, *Using factor oracles for machine improvisation*, Soft Computing **8-9** (2004), 604–610.

[7] Arik Azran and Zoubin Ghahramani, *Spectral methods for automatic multiscale data clustering*, CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Washington, DC, USA), IEEE Computer Society, 2006, pp. 190–197.

[8] Georges Bloch, Dubnov Shlomo, and Assayag Gérard, *Introducing video features and spectral descriptors in the omax improvistaion system*, International Computer Music Conference '08, 2008.

[9] Darrell Conklin, *Music generation from statistical models*, Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences, 2003, pp. 30–35.

[10] Anthony Don, Elena Zheleva, Machon Gregory, Sureyya Tarkan, Loretta Auvil, Tanya Clement, Ben Shneiderman, and Catherine Plaisant, *Discovering interesting usage patterns in text collections: integrating text mining with visualization*, CIKM '07 (New York, NY, USA), ACM, 2007, pp. 213–222.

[11] Shlomo Dubnov, Gérard Assayag, Olivier Lartillot, and Gill Bejerano, *A system for computer music generation by learning and improvisation in a particular style*, IEEE Computer **10** (2003), no. 38.

[12] Nicolas Durand, *Apprentissage du style musical et interaction sur deux échelles temporelles*, Master's thesis, ATIAM, UPMC-IRCAM, Paris, June 2003.

[13] Alexandre R. J. François, Elaine Chew, and Dennis Thurmond, *Mimi a musical improvisation system that provides visual feedback to the performer*, 2007.

[14] ______, *Visual feedback in performer-machine interaction for musical improvisation*, NIME '07: Proceedings of the 7th international conference on New interfaces for musical expression (New York, NY, USA), ACM, 2007, pp. 277–280.

[15] Stefan Kurtz, Enno Ohlebusch, Chris Schleiermacher, Jens Stoye, and Robert Giegerich, *Computation and visualization of degenerate repeats in complete genomes*, Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, AAAI Press, 2000, pp. 228–238.

[16] AT&T Research Labs, *Graphviz - graph visualization software* [online], http://www.graphviz.org/.

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

[18] Tamara Munzner, *Exploring large graphs in 3d hyperbolic space*, IEEE Comput. Graph. Appl. **18** (1998), no. 4, 18–23, http://graphics.stanford.edu/papers/h3cga/.

[19] Emilie Poirson, *Simulation d'improvisations à l'aide d'un automate de facteurs et validation experimentale*, Master's thesis, ATIAM, UPMC-IRCAM, Paris, July 2002.

[20] Stan Ruecker, Milena Radzikowska, Piotr Michura, Carlos Fiorentino, and Tanya Clement, *Visualizing repetition in text* [online], http://www.chass.utoronto.ca/epc/chwp/CHC2007/Ruecker_etal/Ruecker_etal.htm.

[21] Ben Shneiderman, *Inventing discovery tools: combining information visualization with data mining*, Information Visualization **1** (2002), 5–12.

[22] Martin Wattenberg, *The shape of song* [online], http://www.turbulence.org/Works/song/index.html.

[23] ______, *Arc diagrams: visualizing structure in strings*, Information Visualization, INFOVIS 2002, 2002, pp. 110–116.