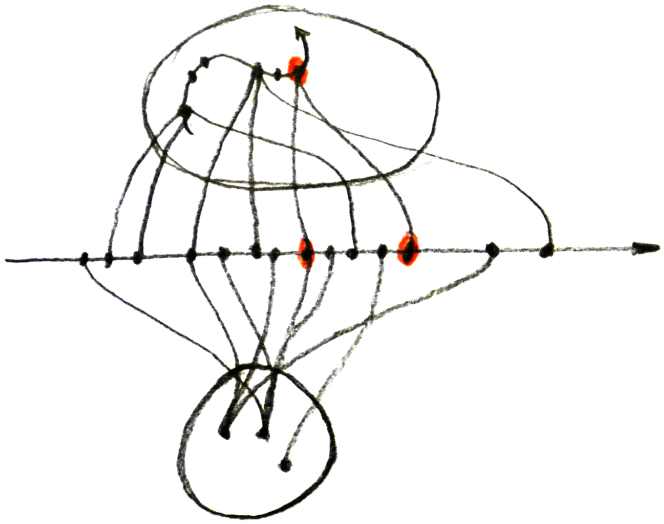
An update on the SOMax project

Laurent Bonnasse-Gahot

September 2014



– internal report –
– not to be publicly distributed –

Contents

1 OMax in a nutshell

This section offers a brief description of the OMax system. See Assayag and Dubnov (2004); Assayag et al. (2006); Assayag and Bloch (2007); Lévy et al. (2012) for more information. The goal here is to focus on the key elements of interest that will allow for a quick and simple understanding of the rest, as we will build upon that.

1.1 Introduction

The OMax software aims at generating musical improvisations that reuse existing external material. It builds a model of the playing of a musician as it captures it, which simply involves the detection of common patterns within the musical material. This analysis serves to find new routes across this musical corpus. The so-called improvisation then consists in a navigation within this structure that both follows the original paths (*ie* replay the original sequence) and, at times, ventures into those new passages, thus jumping to new location, and thereby providing a new version of the captured material. It can be seen as an advanced version of the Musikalisches Würfelspiel, often associated with the figure of Mozart, which were popular dice games in the eighteenth century that proposed to generate music from random concatenation of precomposed musical material. Here this musical material is either taken from already existing source or captured from the live recording of a musician's playing. Moreover, concatenation is not random, but based on the Markovian properties of the sequence itself, which replaces the wit of the composer in composing musical chunks and providing the rules of combinations that will yield a satisfying result.

Within this framework, improvising thus amounts to recombine existing material in a way that is both coherent with the sequential logic of this material and so that it actually provides something different than a mere repetition of the original material while keeping with its statistical property. In contrast with other generative approaches, the system does not construct a model that is eventually independent of the material that was used to train it. Here, the model is directly on top of the data, providing a way of navigating through it in a relevant way. One way of seeing this is to consider that some fine-grained aspects of the musical stream are somehow too complex to be modeled, but will be preserved – to a certain extent – when rereading this musical material. This is exemplified in the use of audio content by the OMax system. While it models the audio sequence by some specific representation (eg. pitch, spectrum, etc), these descriptions account only very partially for

the sound as it unfolds over time (eg. detailed dynamics, articulation, phrasing). A system that would generate musical content using exclusively pitch information would seem pretty bland compared to it. This particularity of the OMax system has been one of its great strength, making it possible to offer a very high sound quality, realistic sound conditions, creative environment as witnessed by the numerous high quality artistic collaborations of the past few year [1]. Of course, it has to be recognized that the strength of this method is also a downside somehow. As said before, improvising is somehow reduced to a smart cut and pasting of pre-existing material, which is very severe reduction, very simplistic modeling of what improvisation is understood as a human skill. Yet, this aspect of the OMax constitutes the fundamental basis of the project, and all the following attempts at exploring and extending the model are done with this framework in mind, as a fundamental condition.

1.2 Description

In brief, the processing chain is the following one. The musical stream is first segmented into discrete units (called *slices*). To fix ideas, if we consider an audio stream consisting in a solo monophonic instrument, this segmentation can be for instance based on the onset of each note (which would produce MIDI-like segmentation). The next step is to label each slice (this assumes that such a classification exists, which requires the definition of a certain metric and of equivalence classes). The final step of this analysis part eventually consists in analyzing the resulting string of symbols so as to find the recurring patterns across the whole musical material. In OMax, this is done thanks to the Factor Oracle algorithm, which was developed by Allauzen et al. (1999) (now actually taking advantages of the algorithmic improvements subsequently introduced by Lefebvre et al., 2002).

This common pattern analysis serves as the basis of the generative process. By navigating this structure thanks to the Suffix Link Tree (Assayag and Bloch, 2007), one is able to connect any location within the musical material of interest to any other location that has a common suffix (called *context* thereafter) (Fig. 1). Reading this structure state by state amounts to playing back the original musical sequence, but navigating through it while allowing jumps (going from one state to another state different from the following one) generates a musical sequence that is both different from the original one and coherent with its internal logic, at least from the point of view of the segmentation and the labels that were used.

[1] Some videos are available at `http://www.dailymotion.com/RepMus`

Figure 1: Illustration of the context switching mechanism. The black line depicts the musical material, represented as some parameter that evolves over time. The red target depicts current location within the musical memory, while the pink ellipses represent the common suffixes found across the musical material. At one point in time, one can either continue reading the original sequence or 'jump' to the marked locations.

1.3 Limitations

Although OMax records the playing of a live musician in real-time, it does not have any listening skill. This capture is only used in order to learn material on the fly, which then serves as the basis for its own improvisations. Yet, when actually generating, OMax does not listen to the current environment. In other words, its choices are only based on internal considerations. Moreover, OMax ignores and breaks pulse feeling. When the original material contains a pulsation (a beat), the improvisation generated by the machine does not convey this pulse anymore, as each recombination, by ignoring this information, break this feeling. And as it does not listen, OMax is not able to synchronize with any external stream.
The SOMax project actually stemmed from these shortcomings – the initial motivation was to provide the machine with some listening skills.

2 An overview of the SOMax environment

2.1 Introduction

The primary goal was to give OMax some listening abilities that would make it more reactive to the current musical environment, both in terms of melodic understanding, so as to harmonize or provide some accompaniment to a monophonic stream, harmonic listening, so as to make a chorus, and rhythmic abilities, so as to synchronize in real-time with live musicians. This section provides a summary of the key concepts and elements that were introduced in the SOMax project in order to provide solutions to those problems. The reader is referred to previous technical reports for more detailed information about the model and the corresponding Max/Msp code.

2.1.1 General idea

With OMax, navigation through the musical memory depends chiefly on the length of the common context. Here, the goal is to guide the navigation so as to take into account the playing of the musician in real-time. First, during the analysis stage, the musical stream is not only analyzed in terms of the detection of common patterns across the sequence, as done before, but is also annotated with different relevant descriptions that will be used for matching with the current live situation. For instance, a monophonic solo might be annotated with the corresponding original harmonic description it was played upon, which will then be used as a way to match the live harmonic situation.

Let us assume that we have arrived at some location within the musical memory. Some actions (for now either the end of the current slice or the detection of an outside event) triggers the computation of a new state within the musical memory. This new state is chosen among the possible states that are given by the oracle structure, following OMax principle (see Section 1), ensuring that the internal logic of the sequence is respected (*ie* according to its Markovian continuity). Instead of simply choosing this state according to the length of its common context, states are also evaluated against one or several external conditions. For example, one might require that the chosen state contains some specific pitch value, or that its harmonic context matches the external one. At the end of this evaluation, we enter a selection phase where one state is picked out according to this evaluation. This solution is then played, either at the time is was first planned to be played, or at a slightly corrected time so as to satisfy some rhythmic condition.
In brief, the processing chain leading the search for a new state is as follows: set of possible jumps → filtering and evaluation of solutions → selection of a solution → choice of an appropriate playing date → playing of that solution.

2.2 Summary of the main features

Most of the material presented in this section comes from previous internal report `sor2reportD1.1.2wp4`, and is reproduced here for ease of reading.

2.2.1 Segmentation and sequence representation

Musical material from the corpus is segmented into different phrases thanks to the detection of silences (Fig. 2). Rests, defined as silences lasting at least 1000 ms, are detected, and used as a symbol that is taken into account when analyzing the whole musical sequence. This make it possible to define places

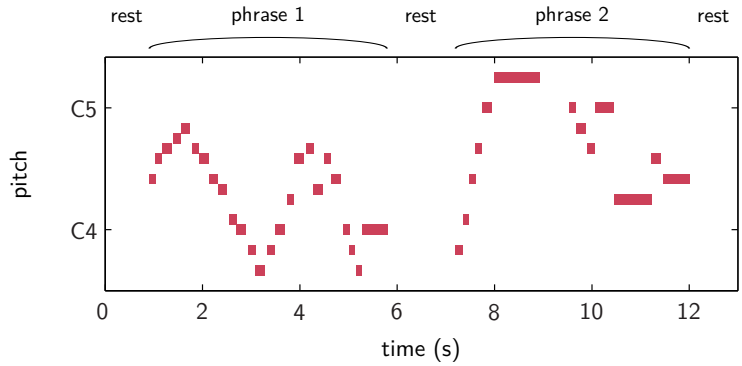


Figure 2: Example of phrase segmentation

where to start improvising (any state that follows a rest) and where to stop (any state that precedes a rest). Moreover, rests have an identity on their own, so they can easily be manipulated, such as being shortened. When navigating the musical memory (*ie* improvising), rests can also be avoided, making the agent more talkative, or conversely be favored, which might be used to end up the musical discourse properly.

Musical sequences are described and analyzed using an interval-based representation allowing transposition. The purpose of using transposition is twofold. First, it makes it possible to manipulate musical material within different context than the one it originally defines, and thus to satisfy (either pitch or harmonic wise) constraints that would otherwise be impossible to satisfy. Second, when analyzing the musical sequence, interval-based representation makes it possible to find similar pattern up to some transposition factor. Recombination possibilities are then strongly increased. Using the Beethoven's late string quartets as a corpus, Fig. 3 represents the average number of potential recombinations (given any position within the corpus) as a function of the length of the context, which is an important factor of the recombination quality. One can see that the use of transposition (red) considerably increases the number of possible recombinations. This is particular important here given that navigation within the musical memory can be highly constrained (if we have more recombination possibilities, chances that constraints are satisfied also increase).

2.2.2 Pulse preservation and synchronization

When learning from a metrically annotated corpus or from a live input with beat recognition, each state in the model (each segmented musical event) is annotated with its beat phase, that is its position within the beat structure.

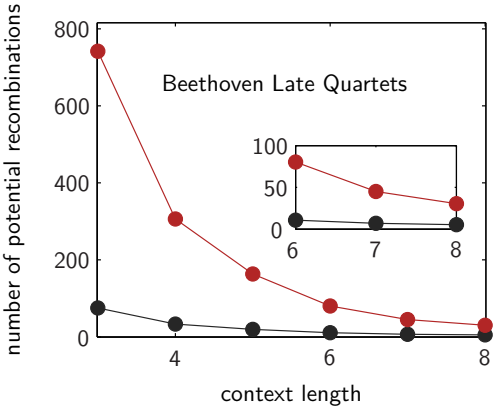


Figure 3: Benefit of interval-based representation

When generating in a context where there some notion of pulse is defined, states with an original beat position close enough to the current one in the ongoing pulse stream, can be favored. This process captures rhythmic/melodic-harmonic correlations in the learned corpus and adapts these correlation in the actual live interaction. When coupled with a mechanism that adjusts the playing date of each state so as to respect its original micro-timing, the music generated by the agent can preserve the pulse feeling and the "swing". This also allows for pulse synchronization. Each agent can be fed with a bang that indicates the current pulse. Thanks to a beat-tracker module that extracts and follows in real time the pulse implied by the playing of a live musician, each agent is thus able to synchronize its internal clock with an external input, either midi or audio. Note that the beat-tracker module can also be used so as to annotate corpora that lack beat phase information (notably when capturing musical stream live).

2.2.3 Soft Harmonic Context

In the early prototypes, the playing of an agent was either rather free, simply recombining while possibly synchronizing to an external pulse, or actually quite constrained, trying to match by inclusion the input pitches to the harmonic and textural content of the states in the model (pitch-inclusion constraints). Although interesting, these latter constraints are of all-or-nothing' type, in the sense that inclusion is either satisfied or it is not. But there was a need for exploring the 'in between' zone through softer interactions between the machine and the musician. Moreover, early modes would not allow the generation of a melody against some accompaniment (played by the user). The idea was to be able to analyze horizontal (melodic) movement while

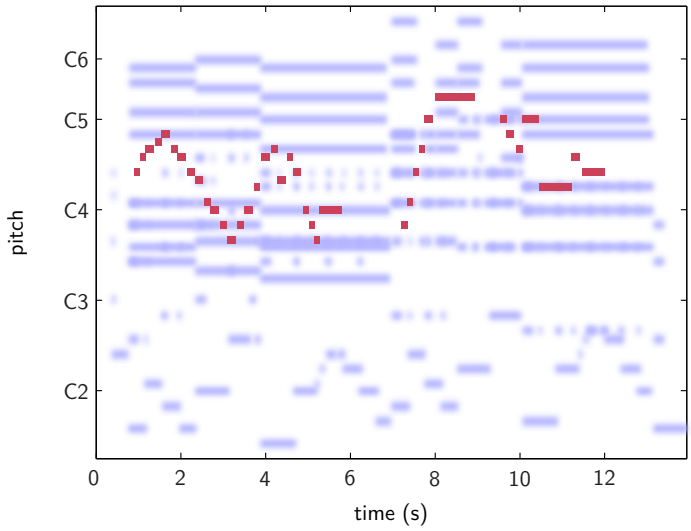


Figure 4: Example piano roll of foreground vs background musical material

also capturing vertical (harmonic) information. Each part can actually be as complex as one wishes (i.e. solo, chords, complex polyphonies), but we will keep with the solo vs accompaniment example so as to make things clearer in the explanation. In the example depicted by the piano roll shown in Fig. 4, the red stream corresponds to the improvised chorus we want to model. This constitutes the musical content that will be played back when generating. The blurred blue stream behind it constitutes the background, and this will be used to define the harmonic context of the solo.

This harmonic background is first used to annotate the solo with harmonic information that will then be exploited as a reference when listening to a different (live) harmonic context. It is transformed into a 12-dimensional vector using partials of each pitch, warping of the frequencies, and some leaky integration. This vector, called a chromagram, basically represents the relative energy of each of the 12 pitch classes. Fig. 5 shows the contribution of a pitch to the final chromagram as a function of time during a 2 seconds note. The contribution first increases, up to some point where it starts to reach its maximum value (this simulates durational accent; see e.g. Parncutt, 1994). When the note is released, its contribution does not go to zero immediately, but smoothly after a certain time, as if the note was still resonating in the mind (cognitive memory persistence). This process aims at capturing local harmonic color behind the specific details of the musical realization (see also the work on tonality induction by Toiviainen and Krumhansl, 2003, that

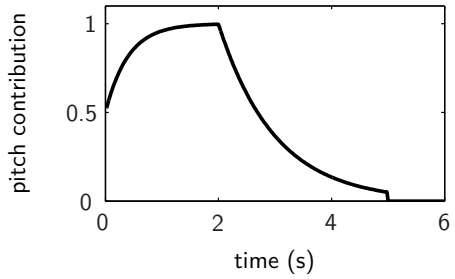


Figure 5: Contribution to the harmonic context of a single pitch that lasts 2 seconds.

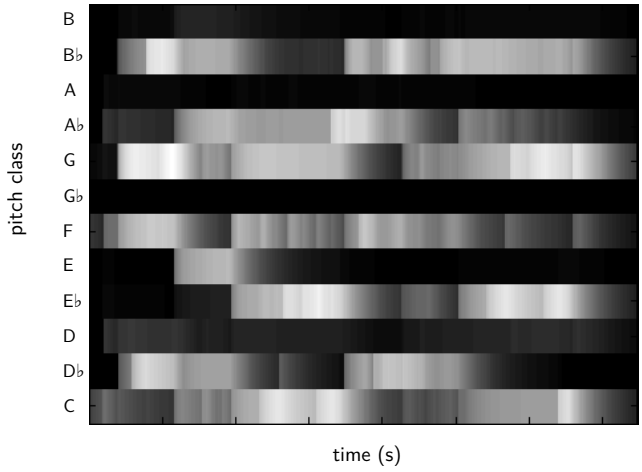


Figure 6: Harmonic context corresponding to the harmonic background depicted in Fig. 4.

uses similar equations and provides experimental support). Fig. 6 presents the harmonic context as a function of time that results from this process applied to the harmonic background of Fig. 4. Within this harmonic space, a certain chord (say C major seventh) occupies a specific location. Within a small radius lie all the different voicings and inversions of that chord. If one increases the allowed radius, one will find common substitutions of that chord, such as A minor seventh or E minor seventh. Note that if current system somehow encompasses this kind of traditional knowledge, it can also adapt to other type of harmonic color.

At generating time, these harmonic annotations are used to guide the navigation within the musical memory. During a live performance, harmonic context can be extracted in real time from both audio and midi input, typically

the accompaniment section. The virtual agent will then try to generate musical content whose harmonic annotations match the harmonic constraints. Similarity (implemented as Pearson correlation) between the chroma used for the constraint and the one in memory (harmonic context) can either be used simply as a preference while navigating or can be set to lie above a specific threshold. The usual accompaniment situation (Solo input, Harmonic generation) benefits from the Soft Harmonic representation, as harmonic context can be extracted from a solo line (or any contrapunctic input) just by the same echoic memory process and chromagram annotation. The input annotation is then matched to the memory in just the same way. Thanks to this non-symbolic representation of harmonic context, and in line with the agnostic approach undertaken by the OMax project, SoMax can adapt to any harmonic content, beyond the use of traditional chords, such as for instance the famous texture of *Les Augures printaniers* from Stravinsky's Rite of Spring (see pedagogical performance Reconstruire le Sacre, leaded by Fabrice Guedy, at Conservatoire à Rayonnement Régional de Paris on Jan 26th, 2013).

2.2.4 The SoMax practical environment

The Max implementation of the SoMax kernel object is pictured in Fig. 7. It can receive different commands that control the object (from start/stop to commands that changes the different parameters of the object), pitch constraints, harmonic constraints and the current pulse. Likewise, it outputs midi content as well as other information that can be used to communicate between agents. The behavior/interaction of each agent can be dynamically programmed, offering new musical possibilities. Finally, it is now possible to learn material on the fly (capturing not only the musical stream itself, à la OMax, but also its harmonic and rhythmic contexts). As SoMax can be instantiated is as many instances as needed, it is possible to have multiple independent agents playing simultaneously while synchronizing with and listening to each other, as well as with external (human) agents. The modular architecture, all-centered around a single versatile object (SoMaxPlayer), allows the user to easily write whatever conductor he/she needs (e.g. a Conductor for a situation where a live musician interacts with three artificial improvisers, 1 mostly harmonic and the 2 others mostly solists, agent A listening to human plus agent B, etc.).
Several Max patch models (called *conductors*) exemplify different typical musical situation – audio (see Fig.8) or midi input, one human / one agent interaction, multi-agent interaction, etc.

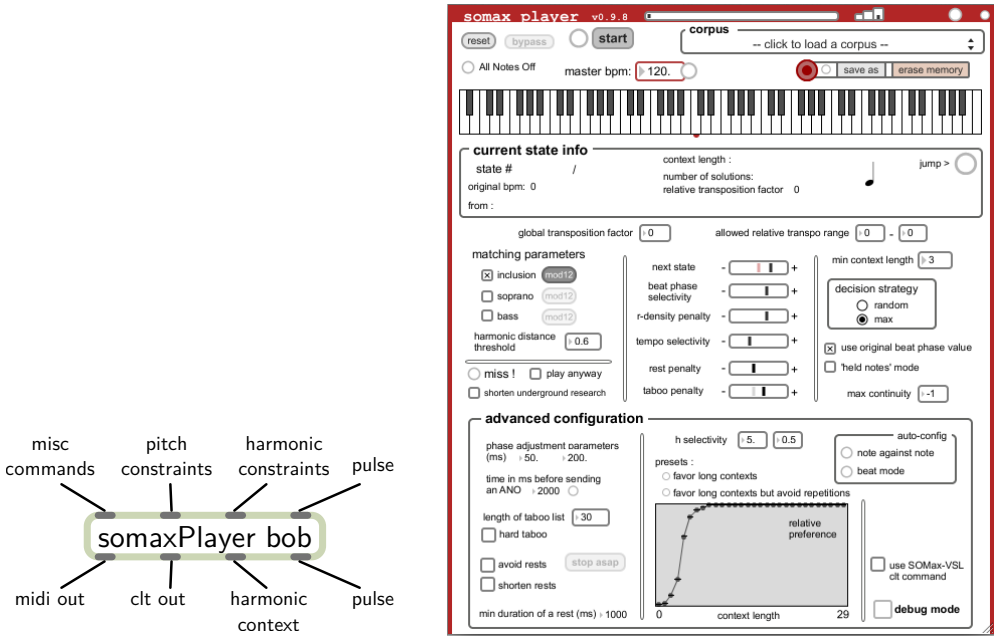


Figure 7: The somaxPlayer object and its basic control interface.

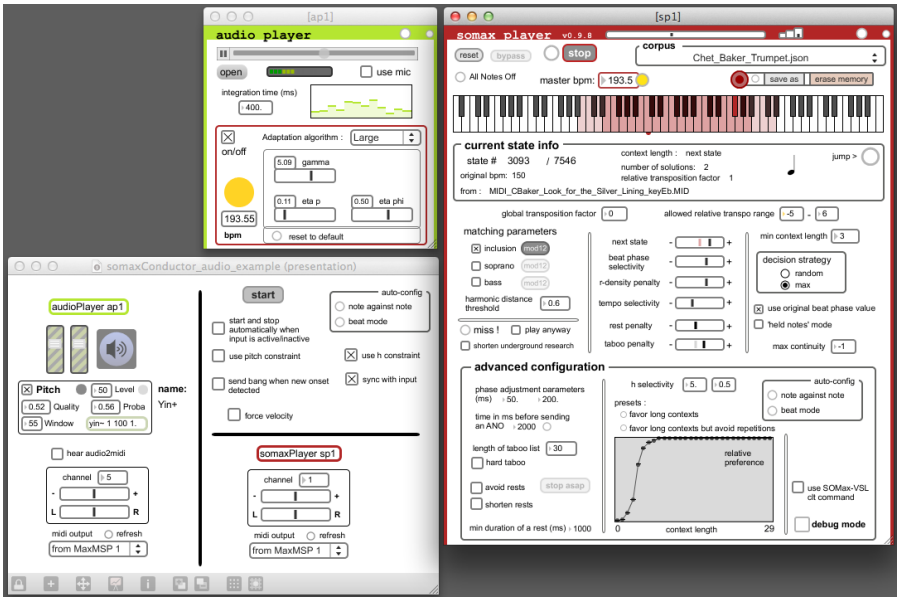


Figure 8: Conductor Interface Patches (left audio input, right Midi Input) for live input harmonic arrangement.

2.3 Limitations

2.3.1 Jumps and context

As we have seen, the musical memory is first analyzed with the Factor Oracle in order to detect repetitions within the sequence. This analysis is then used to navigate within the musical memory, by providing places where to 'jump' that are compatible with current musical generated content, thanks to the common suffix property.
Now, consider the following situation, depicted in Figure 9. Letters here could refer to actual pitches, but could correspond to any other labeling. At the top of the figure, the original sequence, called the memory, used for the recombinations. Below, the generated sequence. Assume that the sequence $\cdots$ A B has already been produced, copied from part I of the original sequence. Then, assume that, for some reasons, notably motivated by the common B element, but not only, the generative process jumps to part II, and the C slice is produced. The generated sequence so far is thus $\cdots$ A B C. Now, consider the possible jumps III and IV. Both are legitimate based on the common suffix property. Yet, within the current framework, taking the point of view of the original sequence itself, solution III will be evaluated as better than IV, for the common suffix is actually longer (D B C), and E would then be subsequently produced. Yet, from the point of view of what has already been generated, *ie* A B C, jumping to part IV is actually a better solution, as the common suffix it shares is actually longer. The element thus produced, F, is actually more coherent with the logic of the sequence. Thus, the best continuation from the point of view of the generated sequence or from the point of view of the original sequence is not necessarily the same one. One should note that this kind of situation was not problematic in the original OMax version, as the continuity mechanism was the only one at work, thereby ensuring that both points of view are actually identical (the fact that each slice has its own label and that this label is preserved during the generation is also important though). But external influences can make that a common situation in the new SOMax improvisation environment.

We will see that the solution we have proposed to this issue will actually bring symmetry in the way internal and external constraints are considered, and both self-listening and listening of outside live musical elements will be handled in a similar fashion.

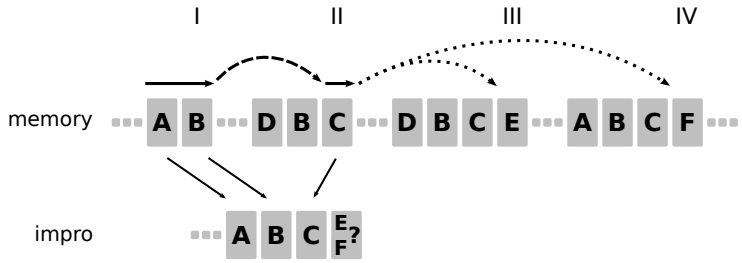


Figure 9: Basic example illustrating the difference between using internal context or generated context during generation of the musical improvisation (see text for details).

2.3.2 Locality of the decision

External influences on the decisions undertaken by the SOMax improviser are always very local, in the sense that they affect current navigation only, but are then forgotten. For example, an external pitch might constrain the musical generation towards states that contain it, regardless of the previous pitches that were played[2]. This information loss is notably responsible for the fact that if you play a melody that is actually present within the musical memory, the machine will usually have hard time finding it and playing the corresponding accompaniment. Note that other difficulties, especially timing and alignment issues, are also responsible for that, but all those questions will be tackled in the next sections.
In the case of the harmonic listening, although influence only concerns the moment of the decision, this information loss issue was dampened by the echoic memory mechanism (see sec 2.2.3) that actually broadened the time span of the decisional influence.
All in all, this calls for some sort of evidence accumulation mechanism that will make it possible to follow and take into account the temporal evolutions of the playing of the live musical environment.

2.3.3 Cartographical blindness

At some point in time, the machine only 'sees' the set of possible jumps, places within the musical memory that stand as potential continuations. The rest of its whole memory is simply ignored at that point. Note that

[2]The first version of SOMax actually considered a slightly more complex mechanism that took the past few pitches that were played into account when evaluating melodic match with current memory location. Yet, this was some sort of a very ad hoc patch-up job working only for pitches.

going abruptly to a region that is not a possible continuation would break the continuity markovian principle underlying OMax. Yet, it might actually happen that it is better to make an unjustified jump and eventually lie in a matching relevant zone than to keep blindly with its own path. We want to have the choice between those two extreme situations (maintaining self internal coherence, following one's own idea *vs* listening, adjusting, reacting to the external influences). There is some sort of an inherent trade-off here, that should be left as a parameter. With the current version of SOMax, the internal logic is maintained, but at the cost of never realizing that there is actually a place that would be a perfect fit to the present situation. In brief, if an external salient sequence of events happen and actually match some places within the musical memory, we want the system to be able to (possibly) take it into account.

3 Exploring, testing, a few new ideas beyond that

This section introduces the few new ideas that were developed during the last months of this project. Those new explorations were directly motivated by the issues raised just above, while keeping with the musical and scientific framework already set by the omax/somax project, that is the creative navigation through a musical memory.

3.1 Mapping and navigating the musical memory

3.1.1 Introduction

Let say you hear the first notes of the Star-Spangled Banner (or any other melody that you actually know). This will automatically activates the memory of that melody, that should then keep singing in your head even in the absence of the external input. This simple phenomenon serves as the basic idea for the present work. The main idea is thus the following. When unfolding over time, a musical stream draws some trajectory within a space that is relevant for the musical situation. Different views can be used to describe this stream: for instance, it can be a sequence of pitch, of mfcc vectors, chromagrams, to name but a few. Any view can be used to listen to either the improvisation that is currently generated by the machine (an internal view) or some external stimuli, such as an harmonic stream. When a element or a sub-trajectory is recognized (a fragment of a melody, a chord progression, or a rhythmic pattern), corresponding portions of the musical

memory get activated. At all times, the machine tries to 'understand' the current musical situation, which comprises both its own playing and the live environment. The most activated parts of the memory points towards parts of the memory that are relevant for the current situation and constitutes material that could be played to fit current musical flow. Again, note that those observations can either come from self-listening of the material that was just generated (possibly anticipating it) or from truly external listening, for example of the melodic playing of a live musician.
Let us reconsider the example depicted in Fig. 9. After stage II, the machine has played A B C. Thanks to self-listening, this A B C subsequence activates the A B C subsequence present in the memory, which can then serve as the basis to orient the improvisation to that part of the memory, thus playing subsequently F, which provides the best continuation here, contrary to a situation where the choice of the continuation would be based solely on the internal representation (D B C), as in OMax and alike. This mechanism will thus provide, by definition, an answers to the problem described in section 2.3.1. Note that it also provides a solution to the cartographical blindness issue (see Section 2.3.3), as anything that can be recognized will be tracked as a potential candidate for a relevant musical case. The whole memory is used when trying to recognize the musical patterns that are currently played, not just the set of the possible continuations as characterized by the suffix link tree.

Of course, brutaly and exhaustively comparing the listened material with the entire memory each time a new observation arrives is not computationnally possible. We will make use of the fact that the memory is to be explored several times, and that intense preprocessing can be done beforehand. Here, we will index each element (or small sequence of elements) of the view under consideration. This will make it possible to quickly access the musical memory and activate the parts that are similar to the listened part. Moreover, by keeping a parsimonious trace of the different places that were activated and by accumulating evidence for places that are relevant sequentially and in time, the process will be able to detect the places in the memory that are relevant to the current musical situation, while taking into account recent past activity, hence proposing a solution to the locality issue described in Section 2.3.2. Finally, the different views will be combine so as to give an activation profile that points to places in the musical memory that best match the current musical flow. Depending on the relevance of each view (how much they get activated), as well as the weights attributed to each view, the machine will be either more prone to respect its internal logic and follow its own idea or match as closely as possible the current external musical situation.

Note that there is an inherent trade-off between the different features that are listened to, and there is not necessarily one 'best' solution. This choice might depend on the will of the user (hence the existence of weights that can be parametrized). But the symmetry in the way the different views are handled (regardless of whether they are external or internal) will allow for a very simple algorithmic treatment of all possible combinations of views that one decides to use.

The following sections describe the process in more detail. Given the limited amount of time dedicated to these explorations, certain decisions were undertaken based on the will of saving time but not on a careful scientific examination, with a clear knowledge of the literature. The goal was to be able to rapidly test the main ideas that we were interested in (handling of time, mixing of views, etc). Each part of the process is quite independent of the others, and should be studied in more depth, improved and expanded.

3.1.2 Mapping and activating the memory

Let us start with a few notations. Let ξ denote some relative time representation, which is a fonction of time t, $\xi \equiv \Psi(t)$. Everything will be internally represented with this MIDI-like relative time. If beat information is present, then the pulse happens at relative time with phase (the decimal part) equal to 0.
The musical memory is defined as a set of 'views', each view consisting in a particular segmentation of the musical stream associated with a particular space of representation (those will be called κ-space hereafter). For instance, one might think of the set of pitches, or the harmonic space defined by a self-organizing map with a toroidal structure trained with western tonal music (see Section 3.5). We will first consider only one view, and then introduce the mechanism that is used to combine the information extracted by the different views.

Mapping.
At time t, after 'hearing' a certain observation o_t (again, this element could come either from the self-generated musical material itself or from outside listening), a particular place κ_i is activated, with activity α_i, that quantifies the similarity between the current observation and the element κ_i. This element could be the quantified pitch value C, for instance. In this case, the activity is either 1 or 0. It could also be a small sequence, such as C D E, with activity simply equal to 1 when matching, and 0 otherwise. More generally, α_i could take any value between, for instance, 0 (no match) and 1 (perfect

match). For instance, here, in practice, when using a self-organizing map of chromagrams as κ-space, this value α_i is related to the Euclidean distance d_i between the normalized chromagrams: $\alpha_i = \exp(-c\,d_i)$, where c is some scalar.

As previously mentioned, scanning the whole corpus each time a new element is observed to find the locations that match would be computationally very demanding. Here, we preprocess the memory in order to store and index all elements (or small sequences of elements, *ie* n-grams) present in the sequence. More advanced indexing methods such as suffix trees could be used. Note that it could also be done using a factor oracle, expanding the one built from the musical memory under consideration. This particular choice was mainly driven by the need of finding a quick and efficient way to access the musical memory, so as to spend time and test the rest of the process described here, but more thought is needed on that part. In brief, the goal here is to directly access relevant material while keeping the computational load manageable, hence the use of n-grams, that act as a pre-filter, when $n > 1$. The value of the n-gram is for now chosen empirically, depending on the size of the stream and the properties of the representational space that is used. The mapping thus constructed allows to locate the k associated places of interest within the musical memory, that translates into the activation of all the events ξ_j, $j = 1 \dots k$, with activity $a_j = \alpha_i$.

Basic activity.
Let us call $\Gamma(\xi)$ the activity at each point ξ of the memory. We define it as the sum of each individual activation:

$$\Gamma(\xi) \equiv \sum_j \gamma_\sigma(\xi;\, \xi_j, a_j) \tag{1}$$

where

$$\gamma_\sigma(\xi;\, \xi_j, a_j) = a_j \exp\left(-\frac{(\xi - \xi_j)^2}{2\sigma^2}\right) \tag{2}$$

represents the basic activation of a single event. The width of the γ function, σ, corresponds to some temporal uncertainty.

Probabilistic interpretation.
At time t, given observations $\mathbf{o}$, the activity Γ can be turn into probabilities thanks to the classic softmax function:

$$P(\xi|\mathbf{o}) \equiv \frac{\exp(\beta\,\Gamma(\xi))}{\sum_{\xi'} \exp(\beta\,\Gamma(\xi'))} \tag{3}$$

where β is some scalar, that will actually receive an interpretation later on. $P(\xi|\mathbf{o})$ can be seen as the probability of choosing ξ as a best match to the current musical flow (again, considering either only internal or external views, or both). Alternatively, this probability can be seen as the probability that everything that has been observed comes from some unknown process that ends at ξ. What follows next in the musical memory is thus a potential candidate for a future match.

Event selection.
In the end, we want to select and play the event $\hat{\xi}$ that offers a best match for the current musical situation, *ie* that maximizes the probability defined above, or, equivalently, that maximizes the activity Γ:

$$\hat{\xi} \equiv \underset{\xi}{\operatorname{argmax}}\, P(\xi|\mathbf{o}) = \underset{\xi}{\operatorname{argmax}}\, \Gamma(\xi) \tag{4}$$

Note that we are not actually interested in any ξ, but one that do correspond to an event, so that the argmax operation is actually performed among the set of events under consideration.

3.1.3 Time and partial sequence matching.

Previous equations have described the basic activation evoked by the recognition in the musical memory of a given observation. Now, in practice, another mechanism is at work which makes it possible to accumulate information over time thanks to the combination of past and present activities.
In the absence of any new observation, activity undergoes a translation in time while decreasing as an exponential function. At time $t + \Delta t$, activity $\Gamma_{t+\Delta t}(\xi)$ thus writes:

$$\Gamma_{t+\Delta t}(\xi) = \Gamma_t(\xi - \Delta\xi) \times \exp\left(-\frac{\Delta\xi}{\tau}\right) \tag{5}$$

where $\Delta\xi = \Psi(\Delta t)$, and τ is a scalar that quantifies the decay of the activity (*ie* the speed of oblivion).

The time translation simply comes from the fact that time keeps flowing. Remember the Star Spangled Banner example formulated above. After hearing the first few notes of the melody, the memory of that melody gets activated. It then keeps running in your head even in the absence of any new input. If one were to look at the location in your musical memory say two beats after the external stimulation had stopped, it would find it situated at, precisely, two beats after.

As for the decrease, one way to see this process is to consider that, in the absence of any new observation, the similarity between the observed pattern and the one in memory (or analogously, the confidence in the relevance of that observation) decreases as an exponential function of relative time. Note that in that case, the probability itself simply translates within the corpus.

Upon arrival of a new observation, final activity results of the combination of the previous activity with the newly-evoked one[3]:

$$\Gamma^{(\text{final})}_{t+\Delta t}(\xi) = \Gamma_t(\xi - \Delta\xi) \exp\left(-\frac{\Delta\xi}{\tau}\right) + \Gamma^{(\text{new})}_{t+\Delta t}(\xi) \tag{6}$$

The scalar τ can be seen as parameterizing the influence of past observations on current activity. Equation 6 will receive a more precise interpretation in terms of probabilities in Section 3.1.4.

Example. Using a simple pedagogical example, we will see how the above-explained mechanism is used to locate places of interest within a musical memory. This will illustrate both evidence accumulation and partial matching features.

Figure 10 presents the score that corresponds to the musical memory. In Figure 11, the x-axis corresponds to relative time ξ, labeled with the respective pitch of each event. We look at the behavior of the activity during the presentation of an input that consists of the three consecutive quarter notes A B C (again, this 'input' can either come from self-generated material, which would correspond to a situation of self-listening, or from external live playing, corresponding to a situation of melodic listening). Here, this example is constructed so that the indexing is made on each individual pitch. Thus, after observing the first A, all the A's in the memory get activated (see Fig 11 top). Then, one beat later, B appears, which entails the activation of all the B's. This activity adds up with the previous one to give the profile depicted in Fig 11 middle. Finally, one beat later C is presented, which yields the situation presented on Fig 11 bottom. It shows that the place that receives the greatest activity actually corresponds to the sequence A B C. Then the next best match corresponds to the subsequence B C. Finally, A D C is evaluated as being better than plain C, given the past pitch and rhythmic match of A D C with A B C, contrary to a situation where only the last C would match (the rightmost peak). The video `activity_dynamics.mp4` enclosed in

[3]Note that, in practice, a ceiling function is applied to final activity so that, when self-listening, current location does not get too much evaluated compared to other possible locations, given the improvisatory context we have chosen.

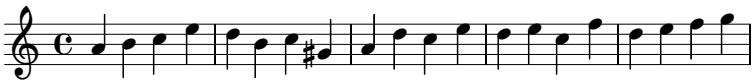


Figure 10: Simple melodic example: score.

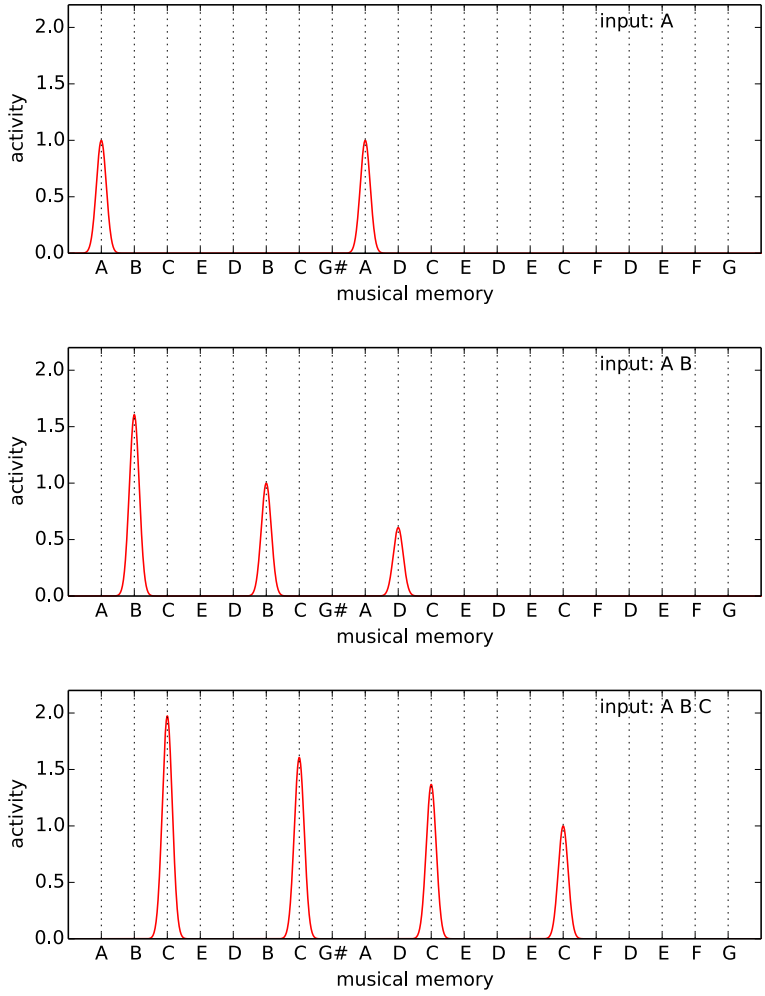


Figure 11: Simple melodic example: sequence matching.

the `files` folder shows the same example but dynamically.

3.1.4 Combining views

We have seen so far the way one stream view is handled, in particular with respect to time. One of the main goal and interest of the present investigation is to be able to combine two or more representations (such as harmony and melody, or pitch, rhythm and spectrum, or, ...). Note that here the mechanism will be all the more interesting as it does not assume the different views to share the same segmentation, or even to be aligned. Completely different views, yet relevant for the musical situation at hand, can thus be simultaneously taken into account.

Let us consider a case involving two views, corresponding to observations $\mathbf{o}^{(1)}$ and $\mathbf{o}^{(2)}$, and with respective activities Γ_1 and Γ_2 (generalization to more than two views is straightforward). In line with Section 3.1.2, we want to find the event that best matches the current musical flow, *ie* that maximizes $P(\xi|\mathbf{o}^{(1)}\mathbf{o}^{(2)})$. Following Bayes' rule, this probability can be written as:

$$P(\xi|\mathbf{o}^{(1)}\mathbf{o}^{(2)}) = \frac{P(\mathbf{o}^{(1)}\mathbf{o}^{(2)}|\xi)P(\xi)}{P(\mathbf{o}^{(1)}\mathbf{o}^{(2)})} \tag{7}$$

Assuming independence and conditional independence given ξ of the observations $\mathbf{o}^{(1)}$ and $\mathbf{o}^{(2)}$, we can write:

$$P(\xi|\mathbf{o}^{(1)}\mathbf{o}^{(2)}) = \frac{P(\mathbf{o}^{(1)}|\xi)P(\mathbf{o}^{(2)}|\xi)P(\xi)}{P(\mathbf{o}^{(1)})P(\mathbf{o}^{(2)})} \tag{8}$$

Using Bayes' rule one more time yields:

$$P(\xi|\mathbf{o}^{(1)}\mathbf{o}^{(2)}) = P(\xi)^{-1}P(\xi|\mathbf{o}^{(1)})P(\xi|\mathbf{o}^{(2)}) \tag{9}$$

Assuming a uniform prior, we can write $P(\xi)^{-1}$ as a constant K. Taking the log of Eq. 9 and making use of Eq. 3 to views (1) and (2) thus gives:

$$\begin{aligned} \log P(\xi|\mathbf{o}^{(1)}\mathbf{o}^{(2)}) \quad &= \quad -\log K + \beta_1\Gamma_1(\xi) - \log\left(\sum_{\xi'}\exp\left(\beta_1\,\Gamma_1(\xi')\right)\right) \\ &\quad +\beta_2\Gamma_2(\xi) - \log\left(\sum_{\xi'}\exp(\beta_2\,\Gamma_2(\xi'))\right) \end{aligned} \tag{10}$$

Given that we are only interested in the argmax, we finally have that:

$$\log P(\xi|\mathbf{o}^{(1)}\mathbf{o}^{(2)}) \propto \beta_1\Gamma_1(\xi) + \beta_2\Gamma_2(\xi) \tag{11}$$

ie

$$\underset{\xi}{\operatorname{argmax}}\, P(\xi|\mathbf{o}^{(1)}\mathbf{o}^{(2)}) = \underset{\xi}{\operatorname{argmax}}\, (\beta_1\Gamma_1(\xi) + \beta_2\Gamma_2(\xi)) \tag{12}$$

In other words, considering total activity Γ as the weighted sum of the two activities Γ_1 and Γ_2 is actually motivated by Bayesian considerations. Note that this is reminiscent of multimodal cue integration (see e.g. Landy et al., 2011; Fetsch et al., 2012). The weight given to each of the views can be seen as the confidence in the relevance of the corresponding view.

Example. To illustrate the process, let us consider the example depicted in Fig.12. At a given time t, specific activities Γ_1 (top, first row) and Γ_2 (top, second row) and total activity Γ (bottom) are shown (the whole reconstructed activity is presented, as a function of relative time ξ). The weights are set equal ($\beta_1 = \beta_2 = 0.5$).
First activity Γ_1 corresponds to self-listening of the musical stream that is currently generated by the machine, hence the highest peak of activity, which corresponds to current location in the musical memory; the other smaller peaks correspond to places where to jump so as to drive the improvisation to another place within the musical memory. The second activity, Γ_2, corresponds to another view which is influenced by external listening, let say some harmonic listening. Here, places that are activated correspond to locations within the musical memory that are relevant to current external musical situation (a live musician playing). Finally, the combination of these two activities yields the resulting activity Γ. The highest peak still corresponds to current location within the musical memory, due to self-listening (hence 'perfect' matching). That would mean in practice that the machine would keep on with its musical idea. Yet, by decreasing the activity of the current location in order to encourage new directions in the improvisation (see Section 3.4), or simply by setting the weights so as to favor external listening, the improvisation would jump to a place that both follows the current self logic and matches the musical situation at hand (note the peak between abscissa 7 and 8).

3.1.5 Algorithmic implementation

The mechanism that has been presented so far might look algorithmically quite demanding, as one needs to keep a trace of all the possibly relevant places in the musical memory, that, if not necessarily useful right now, might happen to be useful in a few minutes. Now, a strong constraint of the project is to be able to respond in 'real-time' to changes in the environment, *ie*

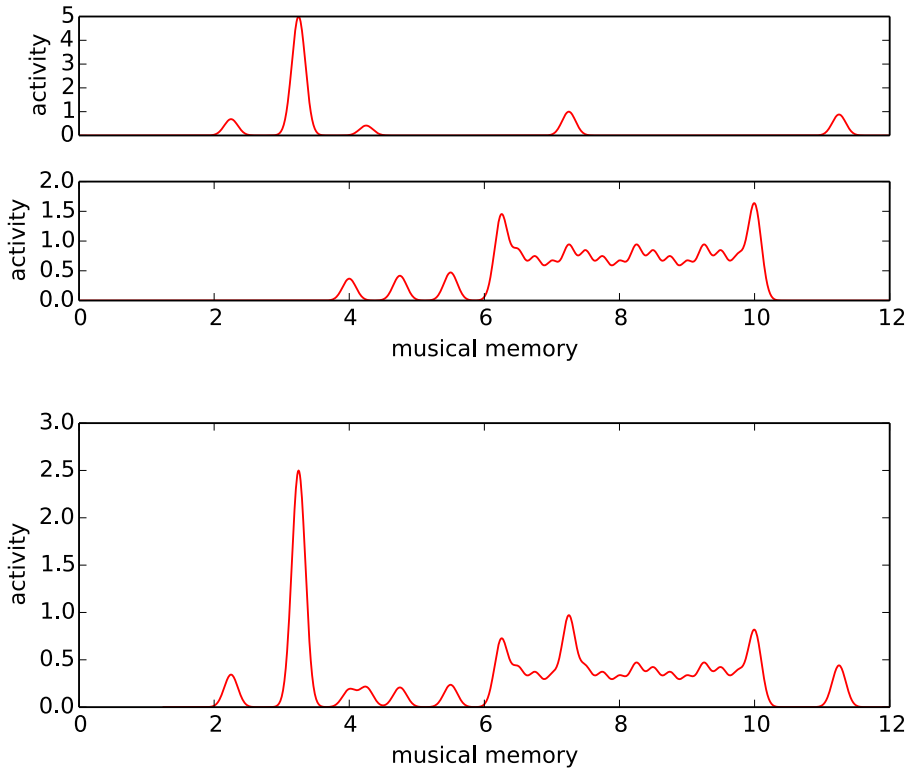


Figure 12: Example of the combination of two views.

quickly enough so that it feels like almost instantaneous. This imposes to use a process that does not take too long to take a decision. Although there is a inherent load due to the fact that one considers all possible solutions at the same times (but the n-gram prefiltering helps to make that manageable), the algorithmic implementation is actually not as complex as it might look, thanks to the particular properties that are explained below.

Parsimonious representation. The activity Γ is represented as a list of basic activities, in the form of pairs (ξ_i, a_i), which is sufficient to recover the whole Γ, simply knowing the γ_σ function (σ comes as a parameter). From a given activity, one can easily compute the activity at any later moment by simply translating the ξ_i's and multiplying the a_i's by the corresponding exponentially decreasing factor (see Eq. 5).

Threshold for minimum activity. If new places were detected without removing ancient traces, the list of possible locations would always keep growing. As the exponential function $\exp(-\Delta\xi/\tau)$ tends to zero as $\Delta\xi$ increases, there would be many locations with a non-relevant very low activity a_i. A threshold is thus introduced so that when any activity a_i fall below it the corresponding place is forgotten (*ie* the corresponding ξ_i is removed from the list of tracked places).

Exponential form and oblivion. The basic exponentiation identity $(\exp(x+y) = \exp(x)\exp(y))$ is copiously used in order to simplify the computations. Thanks to this property[4], two basic activities that coincide in time and finally merge into a single activity can equivalently be stored as a single peak, thus forgetting the details of its origin, while conserving the same information. Indeed, consider the following example. Assume that (1) at time t_1 ξ_1 is activated, (2) at time $t_2 > t_1$ ξ_2 is activated, and that (3) $t2 - t1 = \Psi(\xi_2 - \xi_1)$ (in other words, each element has been recognized, and their succession in the memory matches the time interval between them as they were played). At a time $t > t_2$, both activation then points to the same location $\xi = \xi_1 + \Psi(t - t_1) = \xi_2 + \Psi(t - t_2)$. At this location, the total activity Γ thus writes as:

$$\Gamma(\xi) = a_1 \exp\left(-\frac{\xi - \xi_1}{\tau}\right) + a_2 \exp\left(-\frac{\xi - \xi_2}{\tau}\right) \tag{13}$$

[4]The exponential form was actually chosen here because of this very property.

Using the exp property just mentionned:

$$\Gamma(\xi) = a_1 \exp\left(-\frac{\xi - \xi_2}{\tau}\right) \exp\left(-\frac{\xi_2 - \xi_1}{\tau}\right) + a_2 \exp\left(-\frac{\xi - \xi_2}{\tau}\right) \tag{14}$$

Finally, setting

$$\widetilde{a_2} \equiv a_1 \exp\left(-\frac{\xi_2 - \xi_1}{\tau}\right) + a_2 \tag{15}$$

and factorizing Eq. 14 lead to:

$$\Gamma(\xi) = \widetilde{a_2} \exp\left(-\frac{\xi - \xi_2}{\tau}\right) \tag{16}$$

Instead of remembering both (ξ_1, a_1) and (ξ_2, a_2), it is thus equivalent to remember $(\xi_2, \widetilde{a_2})$. One can 'fuse' two coinciding activities without any loss of information, which help reducing the number of elements to be kept in history.

Approximate fusing mechanism. The above mechanism works when the two basic activities coincide perfectly. But in reality, because of noise in the measurement, in the generation, or simply because external live playing does not match perfectly the data in memory, small deviations represent the common conditions. We thus introduce the possibility of fusing two nearby basic activities into a single one, as described thereafter.

Consider two peaks centered in ξ_1 and ξ_2 with respective activities a_1 and a_2. The sum Γ of this two activities writes as:

$$\Gamma(\xi) = a_1 \exp\left(-\frac{(\xi - \xi_1)^2}{2\sigma^2}\right) + a_2 \exp\left(-\frac{(\xi - \xi_2)^2}{2\sigma^2}\right) \tag{17}$$

We are interested here in a case where the two activities are close enough to be fused. Introducing $\overline{\xi}$ as the center of the resulting peak, one can write:

$$\begin{aligned} \exp\left(-\frac{(\xi - \xi_1)^2}{2\sigma^2}\right) &= \exp\left(-\frac{(\xi - \overline{\xi})^2}{2\sigma^2}\right) \\ &\quad \times \exp\left(-\frac{(\xi - \overline{\xi})(\overline{\xi} - \xi_1)}{\sigma^2}\right) \\ &\quad \times \exp\left(-\frac{(\overline{\xi} - \xi_1)^2}{2\sigma^2}\right) \end{aligned} \tag{18}$$

Assuming that $\overline{\xi}$ is close enough to ξ_1 and ξ_2 (more precisely, $|\overline{\xi} - \xi_1|\sigma \ll 1$ and $|\overline{\xi} - \xi_2|\sigma \ll 1$), and using the above form for both x_1 and x_2, one can expand Eq. 17 in the following way:

$$\Gamma(\xi) = \gamma_\sigma(\xi; \overline{\xi}, \overline{a}) + E(\xi) \tag{19}$$

where

$$\overline{a} = a_1 + a_2 \tag{20}$$

and $E(\xi)$ is an error term whose first order term $E_1(\xi)$ in $\overline{\xi} - \xi_1$ and $\overline{\xi} - \xi_2$ is given by:

$$E_1(\xi) = -\frac{\xi - \overline{\xi}}{\sigma^2} \exp\left(\frac{(\xi - \overline{\xi})^2}{2\sigma^2}\right)\left[a_1(\overline{\xi} - \xi_1) + a_2(\overline{\xi} - \xi_2)\right] \tag{21}$$

and second order term $E_2(\xi)$:

$$E_2(\xi) = -\frac{1}{\sigma^2}\left(\frac{\xi - \overline{\xi}}{2\sigma^2} - \frac{1}{2}\right) \exp\left(\frac{(\xi - \overline{\xi})^2}{2\sigma^2}\right)\left[a_1(\overline{\xi} - \xi_1)^2 + a_2(\overline{\xi} - \xi_2)^2\right] \tag{22}$$

Zeroing first order term yields the expected barycentric value of $\overline{\xi}$ that minimizes the error:

$$\overline{\xi} = \frac{a_1\xi_1 + a_2\xi_2}{a_1 + a_2} \tag{23}$$

The remaining error is maximal at $\xi = \overline{\xi}$. There, it is equal to $\frac{1}{4\sigma^2}\frac{a_1 a_2}{a_1 + a_2}(\xi_2 - \xi_1)^2$. Assuming equal values $a = a_1 = a_2$, if we want this error to be inferior to a certain amount of the activity, $E < \alpha a$, we need to have $|\xi_2 - \xi_1| < 2\sqrt{2\alpha}\sigma$. In practice, we use $\alpha = 10\%$, which means $|\xi_2 - \xi_1| < 0.9\sigma$. By the way, this also gives an interpretation of the σ parameter in terms of temporal resolution.

All in all, in order to simplify the computations, when close enough, we thus replace the two activities (ξ_1, a_1) and (ξ_2, a_2) by a single one $(\overline{\xi}, \overline{a})$ (as defined by Eqs. 20 and 23).

Insertion and sorted lists. Finally, we make use of a sorted time representation. The activity is represented as a list of (relative) time values that is kept in ascending sorted order. The list of new values to insert is directly retrieved in ascending order. Inserting these new values can thus be done in a very efficient way (in a log sense) thanks to a divide and conquer approach.

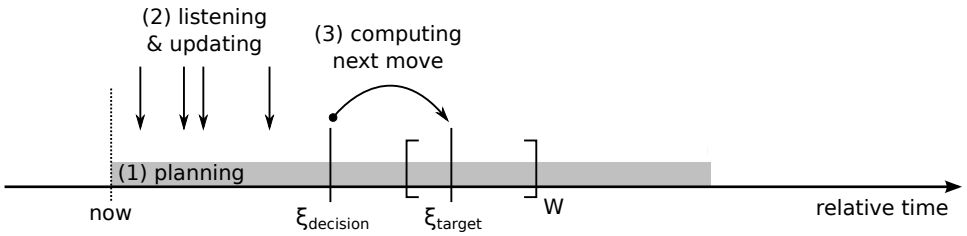


Figure 13: Planning, listening, updating and selecting next location to be played within the musical memory.

3.2 Improvising: planning, updating, and deciding

This section explains in more depth how navigation through the musical memory is done. It is actually very similar to the way it works in previous versions of SOMax (see Bonnasse-Gahot, 2012, §2.3 and §4). One of the main difference with previous versions of SOMax lie in the listening mechanism, and when it is used. Instead of simply taking into account and evaluating outside elements during the selection phase, the system take them into account as they come, updating the corresponding activities simultaneously and independently, while accumulating this information.

Consider the situation depicted in Fig. 13. At current time (now), some event from the musical memory has been played, and, for some time, the machine simply plays what follows in the memory. The machine thus 'knows' what to play next for a certain amount of time: this is what is planned (the grey area in Fig. 13), yet subject to a possible revision, due to the revaluation of the musical situation (which includes, as we have seen, both inner and outer elements). For instance, in the case of audio content, this simply corresponds to filling up a buffer.
Now, some event triggers the computation of a new location within the musical memory, so as to possibly drive the improvisation elsewhere. This change is asked at $\xi = \xi_{\text{decision}}$ and is planned to happen at $\xi = \xi_{\text{target}}$ (this leaves some time to actually compute the decision, delay that depends on the differents parameters involved – size of the corpus, length of the ngrams, notably – but in practice it can be less than 10 ms). In practice two situations are considered. The triggering event either comes from an inner impulse, namely here the anticipation of the arrival of the next event, or from an outside event (see also Bonnasse-Gahot, 2012, §2.3 and §4). In the former case, if desired, the date ξ_{target} when next event is played can be actually adjusted so that its original phase is respected, assuming this adjusted playing date lies within a certain time window W that surrounds ξ_{target} (cf. phase adjustement mech-

anism described in previous reports and in Section 3.3.1). The latter case, usually coupled with a modification of the musical material ('held notes'), serves as the basis for the 'note-against-note' mode, which was used extensively in concert, both with this version and with the previous versions of SOMax (cf. demos). Note that other cases, such as a regular update, could be imagined.
In order to compute the next location, the machine first combines the different activities that have been possibly updated in the meantime thanks to the listening process (as described in Section 3.1.4). The resulting activity is then possibly subject to different modulations, so as to take into account rhythmic pulse influence (see Section 3.3.1), or the other modulations that have been designed (such as taboo for instance, see Section 3.4). Finally, a new location is chosen (which can actually correspond to the one that was planned), and it is played at the expected date, the (possibly adjusted) ξ_{target}. Note that the audio buffer is modified only when it is necessary (modification that is then smoothly handled by a simple crossfade), *ie* when a jump occurs, or when the planned chunk arrives at its end.

3.3 Synchronization, pulse, and more

3.3.1 In a pulsed context

The phase preservation and synchronisation mechanism is directly inherited from previous versions of SOMax (see Section 2.2.2 and Bonnasse-Gahot, 2012). Here, preference for events that have an original beat phase close to the current one in the ongoing musical stream is simply implemented as a multiplicative modulation of the total activity (see Section 3.4). This modulation, depicted in Fig. 14, is described by the following equation: $\exp(\eta[\cos(2\pi(\xi - \xi_{\text{target}})) - 1])$, where ξ_{target} corresponds to the targeted phase, and η is a parameter that control the selectivity of the modulation (the greater η, the greater the selectivity around the targeted phase).

Following this evaluation phase, the playing date of a new state can be adjusted so as to be played at its original phase, thus preserving the microtiming of the original sequence. This is done only if the resulting date of the selected place lies within a certain temporal window (see previous Fig. 13). In the end, such mechanism makes it possible to maintain a pulse feeling when generating a new improvisation, contrary to the classic OMax situation, where the pulse was broken when navigating the musical memory even though it was present in the original musical stream. This also allows for syn-

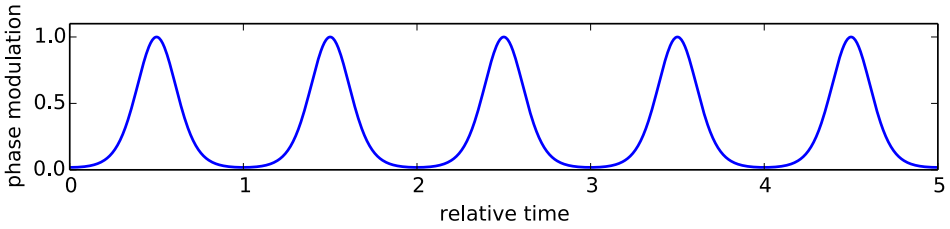


Figure 14: Phase modulation function, $\exp(\eta[\cos(2\pi(\xi - \xi_{\text{target}})) - 1])$. In this illustration, $\xi_{\text{target}} = 0.5$ and $\eta = 2.0$.

chronization to an external pulse, such as the one extracted real-time from the live playing of a musician.

The benefits of this phase preservation mechanism are illustrated in Fig. 15. This example assumes that an external pulse is given, imposed to the system. The corresponding rolling phase is pictured as the continuous light pink curve. In practice, this constraint could either come from an external metronomic indication, or from the output of a real-time beat-tracker. The top figure corresponds to a situation where no phase influence nor adjustment is made. In that case, one can see that each time the improvisation jumps to another place within the musical memory (at dates marked with a downward arrow), the corresponding phase curve is disrupted, which means that the regularity of the conveyed pulse is broken. It is also obviously not able to synchronize with the rhythmic constraint, as in OMax (note though that the classic OMax situation would actually picture a worse scenario, as the speed of the different chunks would not even follow the one imposed). On the contrary, when the phase influence and adjustment mechanisms are employed, the unfolding of the phase is not disrupted, and the improvisation line up nicely with the external rhythmic constraint (see bottom figure).

3.3.2 Beyond pulse: using a sync track

Traditionally, two opposite rhythmic situations are considered. Either a totally free situation, where no rhythmic constraint is taken into account, or a pulsed context, defined by the presence of well-defined pulse. So far, classic OMax has been used in the former kind of situation, as it completely ignores this kind of information at all stages of the process. This has yet given rise to interesting musical situation. The other situation is also of interest, where a well-defined pulse is present in the musical environment. The challenge here is not only to be able to detect the pulse within the live musical context,

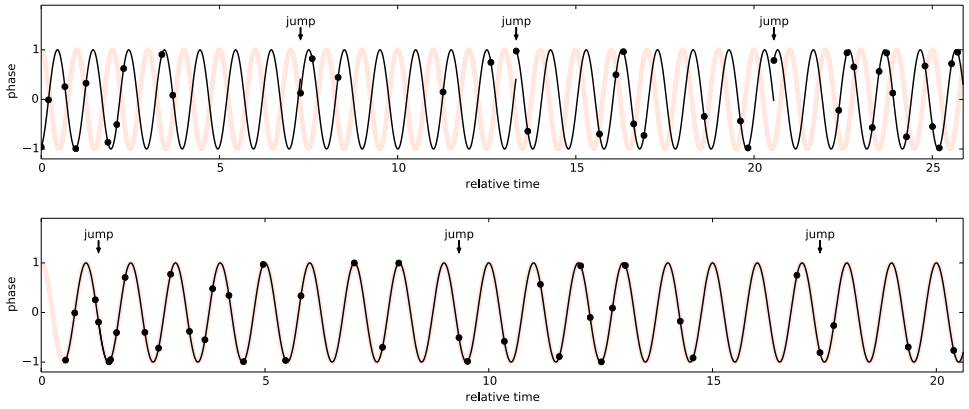


Figure 15: Illustration of the phase preservation and synchronization mechanisms (with (bottom) and without (top) such a mechanism). The light pink curve corresponds to the external imposed pulse, while the black curve represents the inner phase of the improvisation as it unfolds over time. The black dots represent onsets of the notes that are played. Jumps are marked with the downwards arrows.

but also to be able to synchronize to that external pulse. We have just seen how the SOMax prototype has answered this issue, namely a mechanism that maintain the ongoing pulse during generation, in spite of the multiple jumps within the musical memory. This has also been integrated in the most recent versions of mainstream OMax. Note also that an old version of OMax actually contain a 'beat-mode' (Assayag et al., 2006), where the machine was able to synchronize with a regular beat structure, thanks to a regular slicing of the musical material into regular beat chunks. The mechanism presented hereafter works with any kind of segmentation.

Here, we got interested in a situation where one would like to get some level of synchronization without the need to assume that a pulsation actually exists. This could be used in quite free musical context, but would allow for some sort of rhythmic listening and adaptation to the musical environment, beyond pulsed context. In practice, the following exploration was done in a very short amount of time (though *le temps ne fait rien à l'affaire* as Alceste put it), and conceived in a pretty naive way for now, but as it gave rise to interesting musical situations, it is described anyway.

The idea is the following. Given an external list of onsets (the last few notes that have just been played by the live musician), we want to change

the speed of the improvisation generated by the machine so that this new speed would provide a more relevant rhythmic match to the current musical situation. This is done by maximizing the alignment between these external onsets and the pattern of onsets given by current location in the musical memory. It can somehow be seen as a generalization of the pulsed case, which would correspond to the use of a comb of onsets, to cases that involve any type of patterns of onsets, not necessarily regularly spaced. In practice, the track used for synchronization could be a completely dedicated view, or some existing view can be reused for the purpose. The musical applications that were so far undertaken actually made use of the view dedicated to melodic listening. Such a mechanism requires some notion of quality of alignment, which is evaluated for different possible time stretch. The new speed that maximizes the synchronization is then taken as the new speed. By applying this process periodically, the machine adapts itself to the speed of the current musical situation.

We thus need, given two list of onsets, a function that quantifies the degree of alignment. Again, we want to be able to make these computations as fast as possible, given the 'real-time' objectives of our system, and this constraint has driven many of the choices that are presented below.
Before seeing how this function of alignment is used within our context, let us first consider an abstract case involving two lists of onsets, $\{a_i\}$ and $\{b_j\}$. In the spirit of the previous will to give some flesh to an onset (see Eq. 2), we define $A_i(x)$ and $B_j(x)$ as a function of the center of the onsets a_i and b_j, where x lies within the same space as the a_i's and b_j's, and σ represents some sort of time uncertainty:

$$A_i(x) \equiv \exp\left(-\frac{(x-a_i)^2}{2\sigma^2}\right),\ B_j(x) \equiv \exp\left(-\frac{(x-b_j)^2}{2\sigma^2}\right) \tag{24}$$

The degree of alignment, let us call it Q, is quantified as the overlap between the two series of onsets, which is measured as the integral of the product of the two summed respective lists of onsets:

$$Q \equiv \int \left(\sum_i A_i(x)\right)\left(\sum_j B_j(x)\right) dx \tag{25}$$

Assuming σ is small enough compared to the typical distance between two onsets, we then make the following simplification:

$$Q \approx \sum_i \int A_i(x) B_{\widehat{j(i)}}(x) dx \tag{26}$$

where $\widehat{j}(i)$ is defined as the closest element to a_i among the b_j's.

Finally, evaluating each integral product leads to:

$$Q \approx \sigma\sqrt{2\pi}\sum_i \exp\left(-\frac{(a_i - b_{\widehat{j}(i)})^2}{4\sigma^2}\right) \tag{27}$$

In our case, we try to find the best change of speed that would provide a better understanding of the external events with current location in the musical memory. From the current time, we look back within a certain time window to the past few onsets and see how they get align with the corresponding recent past within the musical memory, not only considering current speed, but also a certain range of speed around current one. In other words, we compute $Q(\alpha)$, where α is a stretch factor, for a range of α values. In the end, we are interested in $\text{argmax}_\alpha\, Q(\alpha)$ (provided that it actually corresponds to a peak) and set the speed of the improvisation to this more relevant speed (which is equal to α times the current speed). Note that given the form of this quantity, and once again taking advantage of the fact that the a_i's and the b_j's are given as sorted lists, we are able to make the required computations sufficiently fast for our purposes.

Figure 16 pictures an example of the use of such a synchronization mechanism. The top figure presents the situation before the speed change: the alignment is pretty poor. The middle figure shows the evaluation of $Q(\alpha)$ for α between 0.9 and 1.1. It shows that by reducing the speed of its playing (the optimal α is equal to 0.92 here), the machine should have a speed that is more synchronized with the external situation (in relative time, the external onsets get closer to one another). This new speed results in a much nicer alignment between the live external playing of the musician and the musical memory (see Fig. 16 bottom). Remember though that this alignment is theoretical, as it concerns the immediate past, but it is hoped that it will result in a better rhythmic match in the near future.

3.4 More on modulations

We have seen in Section 3.3.1 that preference for specific beat position was handled thanks to a (multiplicative) modulation of the total activity. This made it possible to find a place that is both relevant for the musical situation at hand and congruent with the beat phase under consideration. Similarly, other type of modulations can be applied to the activity before the decision is reached.

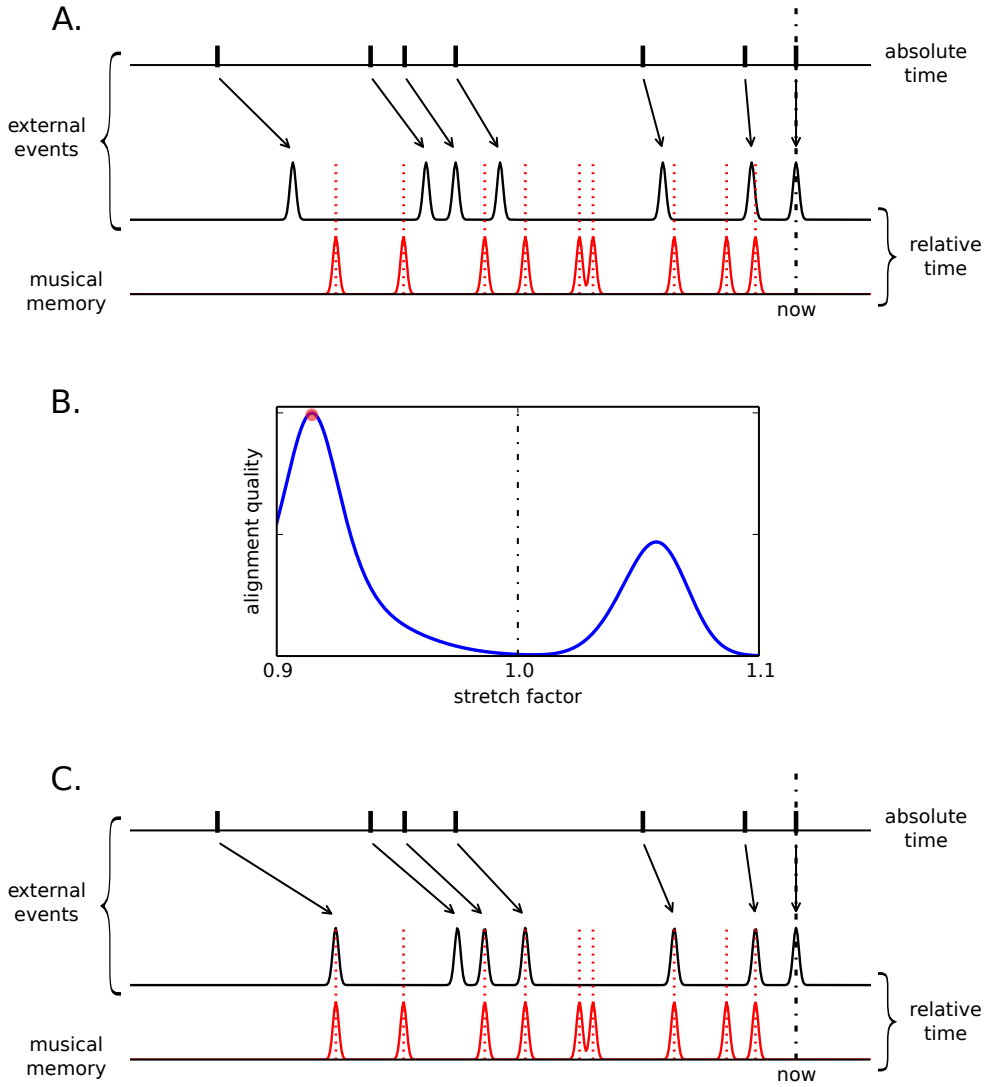


Figure 16: Synchronizing with an external stimuli. (A). Situation with current speed. (B). Evaluation of the degree of alignment for different time stretch values. (C). Situation with best speed change.

As a first example, consider the 'taboo' states mechanism, which was used in previous versions of OMax and SOMax (see Assayag and Bloch, 2007; Bonnasse-Gahot, 2012), which consists in keeping track of the places that were already visited so as to banish them or at least penalize them by decreasing their probability of being selected in the future. This was originally designed in order to prevent the machine from getting stuck in a loop. The same thing is now present in the current version of SOMax, and is simply implemented as a modulation of the states that are in the taboo list. When the multiplicative factor is superior to 1, this consists in favoring those visited places, whereas when it is inferior to 1, it amounts to penalizing these states. Here, we have further experimented by using a time-based taboo, so that events that within a certain time back in the past can be either favored or penalized. This makes it possible to purposely create musical loops that are not only coherent from a musical continuity point of view, but also can rhythmically synchronized with an external groove.

An important modulation is the one that applies to current location within musical memory. In the absence of such a modulation, after playing just a few notes, current location can receive the same evaluation as other positions, which might result in many undesired jumps. Conversely, after a while, current location might get an overwhelming evaluation, since by construction it constitutes a perfect match with what has just been played by the machine: this would result in a mere replay of the original sequence, without any jumps. We thus need a mechanism that can both ensure that the original sequence is read during a minimal amount of time, and that, on the contrary, the improvisation can jump to different places on a regular basis, so as to provide the diversity the system was built for. This was implemented in two ways. In the first possibility, current location is normally favored, thus ensuring a certain continuity, and an external command, which is either automatically called or sent by the user, is used to ask the system to 'jump'. In the second possibility, after a jump has been made, the modulation starts by favoring current location, thereby guaranteeing continuity, and its value decreases over time, so that at some point current location is actually penalized compared to other location, which eventually leads to a jump. This decrease is controlled by some time constant that can be modified so as to adjust the frequency of the jumps, leading to more or less variability. The interest here is that, after a certain minimal amount time, the jump will happen as soon as a 'good' candidate will be found, with a notion of goodness that will decrease over time, so that it will jump eventually in any case. In other words, the more it fails to find an interesting place, the more it lowers its expectations.

Other type of modulations can be easily and readily implemented. For example, a mechanism such as the one present in OMax, that allows the user to control the improvisation by favoring or penalizing certain regions of the musical memory, could be easily added to the current prototype.

3.5 A note on the different representations that were explored so far

Beyond the obvious use of pitch to describe the musical data, (or virtual pitch in a polyphonic case), we have consider different other cases, such as the use of chromagrams (following the technique described in Section 2.2.3), mel frequency cepstral coefficients (MFCCs), or event short melodic contours. Here, given such a descriptor, we furthermore make use of the self-organizing map technique to analyze the musical data. Self-organizing map is well-known neural-network approach (Kohonen, 1995) that precisely aims at making a cartography of the space of the training data, capturing the essential aspects of it in an unsupervised fashion. It has proven to be useful in many domains, especially in music cognition, notably to model harmonic perception (Tillmann et al., 2000; Janata et al., 2002; Toiviainen and Krumhansl, 2003; Janata, 2007), as well as other musical aspects (see e.g. Toiviainen et al., 1998). Its agnostic property fits well with the characteristics of the present project. The algorithm builds a *map*, *ie* a low-dimensional discretized representation, of the data under consideration. Following the training phase, the structure and topology of the resulting map reflect the statistics of the data. This makes it possible to extract a relevant codebook that provides some sort of an elementary dictionary of the most representative elements within the data (see Fig. 17 for an illustrative example). This 'lexicon' is then used to map the musical memory and directly access any element that is similar to the query. Although it could be used directly to map the musical memory, the SOM is actually clustered into a predetermined number of classes (using standard k-means algorithm) that not only reduces the amount of basic shapes to consider, but also allows for the n-gram indexing approach we described before (see Section 3.1.2).
In practice, the mapping of the musical memory can either use a map that is directly built from the musical content under consideration (either MIDI or audio), or use a preexisting map. Here, cases involving self-listening follow the former approach while harmonic listening involves some existing knowledge, using a map previously trained on a large corpus of western music.

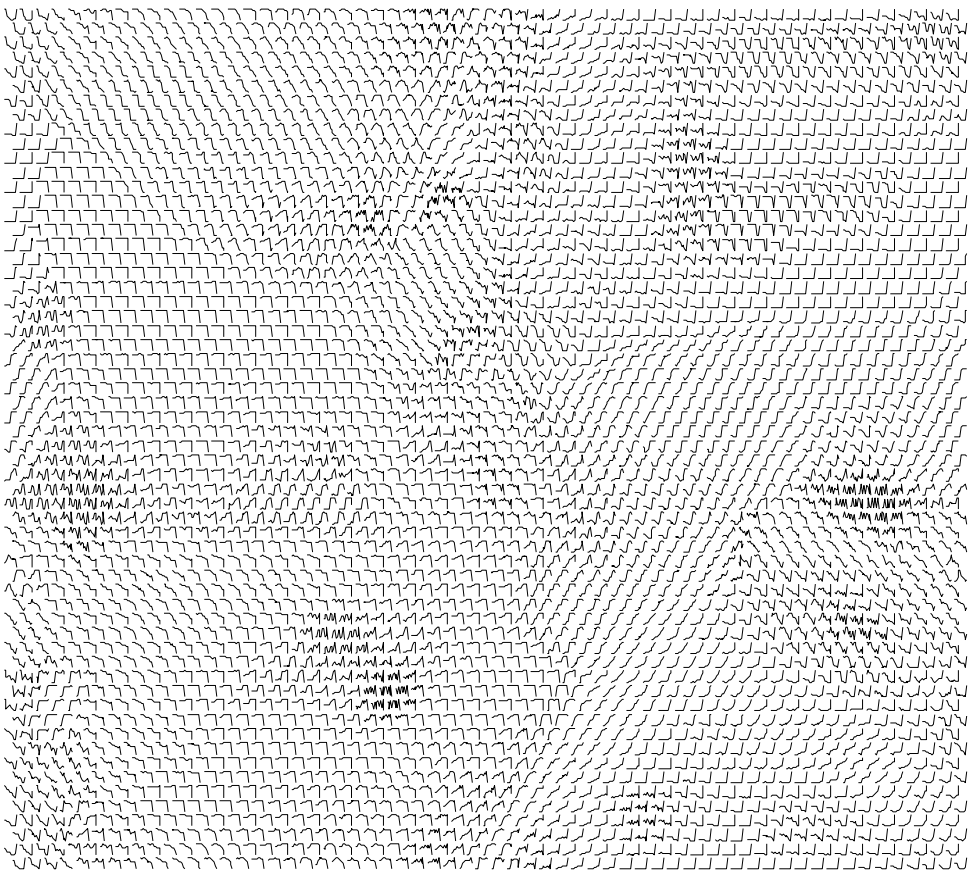


Figure 17: Self-organizing map trained on short melodies extracted from the violin part of Beethoven's late string quartets.

3.6 A few demos

Journées portes ouvertes de l'Ircam, 2014.

This public demo was held on June 6th, 2014, with saxophone player Remi Fox.
The corresponding code is provided in the main soma folder. It exemplifies possible uses of the somax object for specific musical situations. Four situations were considered:

- **demo_1_hiphop** involves one audio virtual agent, trained on a hip-hop piece called 'The thief's theme'. The analysis extracts and annotates beats, use beat-wise frames and mfcc analysis. During the improvisation, the machine first freely recombines this material using the phase preservation mechanism, so that the pulse feeling is maintained and kept steady, even when a jump occurs (every two beats on average). It is subsequently more and more influenced by the harmonic context computed real-time from the live sax solo. In the end, the `auto_bpm` feature is finally turned on, which the musician uses to slow down the improvisation.

- **demo_2_evans** exemplifies the now classic situations of improvised harmonization of a melodic solo (see previous SOMax demos available). The corpus is built from a MIDI file of the piece Time Remembered, composed by Bill Evans, and performed in the style of the famous piano player. The improvisation alternates between the note against note mode and a play mode that features auto-adjustment to the external rhythmic situation. In both cases, melodic listening (mod 12) is used to influence the musical improvisation of the machine. This melodic listening is based on a monophonic line extracted from the original MIDI file and defined as being the highest notes of the polyphonic content.

- **demo_3_remi_clone** features for the first time live recording and on the fly analysis of audio content, *à la OMax*, coupled the multi-agent and listening abilities of the current SOMax environment. The musician first starts alone, and this musical material is recorded and analyzed online. Two virtual clones of the live musician then enter the stage, one after the other, using slightly different parameters to generate different improvisations. They employ both melodic and harmonic listening. The live musician is able to control his virtual doubles by recalling some parts of the previously recorded material played earlier in the same performance. Depending on the parameters, the machine is more or less prone to follow the influence of the live musician. All in all, the global

performance alternates between phases of total convergences, where the three agents (Remi and the two virtual players) play in unison, and phases of heterophony, where each musician get away from each other.

- **demo_4_schoenberg** is the most complex situation of these four demos. It involves two virtual agents, one built from a corpus of Schoenberg's Drei Klavierstücke, Op. 11 (actually using different transposed versions of these pieces), and one that captures and learns audio material on the fly. During the first part of this demo, the Schoenberg virtual player plays while listening melodically (mod 12) to the live musician (again, the melodic line used for listening is built from the highest notes of the MIDI content). Meanwhile, the second agents captures and analyzes the live solo of the player (using two microphones: one for recording the audio improvisation in a good quality, which will then be reinjected, and one, a piezo attached to mouthpiece of the instrument, for the pitch and onset analysis), while annotating this stream with the harmonic content provided by the first virtual agent. In a second part of the performance, the two virtual agents play together, listening to one another. Virtual saxophonist is influenced by harmonic context and virtual Schoenberg is influenced by melodic patterns (mod 12). Finally, the live saxophonist, as well as another musician, percussionist Laurent Mariusse, finally join the stage.

A few small illustrative examples

Three basic examples are provided in the `demos/small_examples` as simple illustrations of the new explorations, and it notably shows an answer the issues raised in Section 2.3:

- `sowhat_loops` illustrates the use of phase influence and adjustments in a pulsed context, as well as the use of the beat-wise taboo with positive modulations that can be used to automatically create coherent synchronized loops.

- `sacre_hctxt` illustrates how the external influence on the musical material can directly be applied at the same surface level than what is heard (harmonic listening here). It addresses the cartographical blindnesss explained Section 2.3.3.

- `bach_inventio` illustrates how the external input can exert influence on a hidden track, which is melodic here. The right hand is played by an external midi player, while the left hand is played by the somax

player. The corpus is built from all the Bach two-part inventions – the left hand is used as the main view, while the right hand is used as the melodic view, used for melodic listening. The result exemplifies the answer provided to both the cartographical blindness and the evidence accumulation issues. It also shows how the rhythmic adaptation mechanism presented in Section 3.3.2 can be used, even within pulsed context, to follow speed variations. Note that the goal here is not to do some score following, though this is the behavior one might expect from such a setting, and it was not possible with previous versions of SOMax

Previous SOMax performances.

Some videos are available at `http://www.dailymotion.com/RepMus`. Four main past SOMax performances can also be found in the `demos` folder:

- Nuit de l'Impro, CNSMDP, Oct 12th, 2012, broadcasted live on France Musique, with Vincent Lê Quang (saxophone)
- Les Rencontres du Numérique de l'ANR, Cité des sciences et de l'industrie, April 18th, 2014, with Carine Bonnefoy (piano) and Remi Fox (saxophone)
- Journées Nationales du Développement Logiciel JDEV, École polytechnique with Carine Bonnefoy (piano) and Remi Fox (saxophone)
- Concert création pour dix-huit musiciens, Carine Bonnefoy, 18 au max, CNSMDP, Oct 23th, 2013

3.7 A note on the delivered code

The different pieces of code can be found in the accompanying folders. This experimental code was developed with the primary goal of testing the few exploratory ideas that have been described in this Section 3. Although it has been used in a real public setting, it is far from being finished, not to say polished. Moreover, despite the fact that the project retained its original name, and that bits of code were recycled, the core of the project is entirely new. Some features that were present in previous versions of SOMax are no longer available here, not because of their lack of interest, but as a result of the lack of time available for the present implementation. Thus, for instance, the interval-based representation, that allowed for transposition and thereby considerably increased the capabilities of the machine to cope

with various situations, is not implemented in the current version. Likewise, the phrase and section segmentation is not exploited at this point, although it provided interesting points where to start, and where to stop (see Section 2.2.1). Nonetheless, the code was developed to a sufficient state where the benefits of the current approach over previous versions are quite clear, notably in the way it has answered the issues raised in Section 2.3. It should be seen as complementing previous efforts, rather than totally replacing them.

Somehow renewing with the organization of the initial OMax prototypes, built around Lisp and Max in order to take advantage of the best of the two programming worlds, the present prototype involves both the Python language as the core for the representations and analyses and Max/Msp as the reactive real-time interface. The binding between the two is achieved thanks to the `pyext` object, developed and freely distributed by Thomas Grill (see below for more technical details). The code is organized as follows. The core of the project is written in Python, and is present in the file `soma.py`. All the fundamental functionalities, such as the mapping of the memory and the handling of the activities, can be found in this file. The audio and midi interactions are handled in Max/Msp. The file `soma_rt.py` makes the bridge between the two. This yields the somax object, which corresponds to a single virtual agent. Following what was done in previous versions of SOMax, this object is not meant to be used on its own, but is to be used within another patch, called conductors here, which will define the way it is specifically used. As before, the behavior of the agents can be fully scripted. Each agent can receive a certain number inputs, corresponding to commands that allows to change the different parameters, to start and stop the improvisations, to change the corpus, to specify the different listening constraints (melodic, harmonic, rhythmic), etc; it outputs both musical content (MIDI or audio) and extra information, such the current internal pulse of the agent, in the form of a bang message. Not much documentation is provided here, but the reader is referred to previous documentations as well as the code itself, that should speak by itself for certain specific need, such as the list of available external commands.
No single patch can match all the possible musical situations one might think of. Should such a patch exist, it would be a Rube Goldberg machine that would do way too much than needed for the current purpose, hence obscuring the use of the environment. Two conductor patches are provided, one to basically deal with a MIDI input, one for audio input (see Fig. 18). Both involves two virtual agents. The patches clearly define both the agents involved and the possible interactions between them. The code used for the *Journées portes ouvertes* is also included, and provides further examples of

the possible uses of the present system. It should be easy to cope with new situations by simply adapting the conductors that are provided.
These conductors should be seen as pedagogical examples of the possibilities offered by the current system. Similarly, the agent itself should also be seen as one particular illustrative instance that belongs to much larger variety of situations. While the code for the handling of the views is quite generic, each player is hard coded as possessing three 'stream views': one for self-listening, one for harmonic listening, and one for melodic listening. Many other situations could be quite readily adapted from this current case. For instance, self-listening might employ more than just one view, as is the case for now. Likewise, new types of view, such as one that consider rhythmic patterns, might reveal useful.
Finally, the Matlab code for creating the databases, both from MIDI and audio files, is also provided.

On a technical side, current version of the prototype has been developed and tested on Mac OS X 10.7.5. It is known to work on Max/Msp 5.1.9, and uses the `py/pyext` object, v0.2.2[5]. The tools to build the corpora were developed on Matlab 7.11 (R2010b), and require both the MIRtoolbox (version 1.4.1 here)[6] (Lartillot et al., 2008) for the audio analysis and the SOM Toolbox (version 2.0)[7] (Vesanto, 1999) for the use of the Self-Organized Maps. The MIRtoolbox can be found in the accompanying `external_libraries` folder, and it actually includes the SOM Toolbox. Finally, the `beatroot` java object[8], developed by Dixon (2007), is also used for the offline audio beat analysis.

[5] `http://grrrr.org/ext`

[6] `https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox`

[7] `http://www.cis.hut.fi/projects/somtoolbox/`

[8] `https://code.soundsoftware.ac.uk/projects/beatroot`

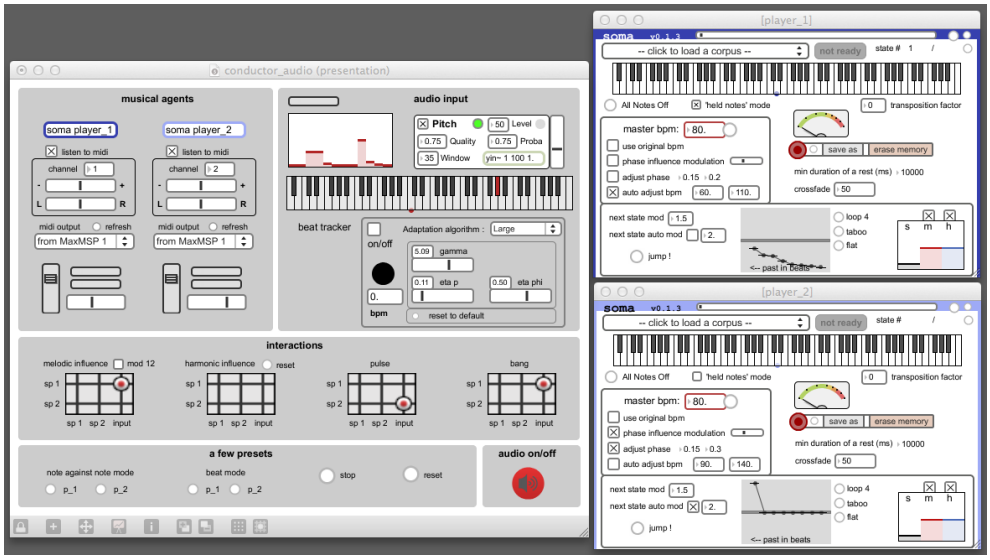


Figure 18: Example of the use of the SOMax player object: the audio conductor.

References

Allauzen, C., Crochemore, M., and Raffinot, M. (1999). Factor oracle: a new structure for pattern matching. In Pavelka, J., Tel, G., and Bartosek, M., editors, *Proceedings of SOFSEM'99, Theory and Practice of Informatics*, Lecture Notes in Computer Science 1725, pages 291–306, Milovy, Czech Republic. Springer-Verlag, Berlin.

Assayag, G. and Bloch, G. (2007). Navigating the oracle: a heuristic approach. *International Computer Music Conference '07 (Copenhagen, Denmark)*, pages 405–412.

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

Assayag, G. and Dubnov, S. (2004). Using factor oracles for machine improvisation. *Soft Comput.*, 8(9):604–10.

Bonnasse-Gahot, L. (2012). Prototype de logiciel d'harmonisation/arrangement à la volée: Somax v0.1. Technical report, IRCAM.

Dixon, S. (2007). Evaluation of the audio beat tracking system beatroot. *Journal of New Music Research*, 36(1):39–51.

Fetsch, C., Pouget, A., DeAngelis, G., and Angelaki, D. (2012). Neural correlates of reliability-based cue weighting during multisensory integration. *Nature Neuroscience*, 15:146–154.

Janata, P. (2007). *Tonal theory for the digital age.*, chapter Navigating tonal space, pages 39–50. Stanford (CA): Center for Computer Assisted Research in the Humanities.

Janata, P., Birk, J., Horn, J. V., Leman, M., Tillmann, B., and Bharucha, J. (2002). The cortical topography of tonal structures underlying western music. *Science*, 298:2167–2170.

Kohonen, T. (1995). Self-organizing maps. In *Series in Information Sciences*. Springer, Heidelberg.

Landy, M. S., Banks, M. S., and Knill, D. C. (2011). Ideal-observer models of cue integration. *Sensory cue integration*, pages 5–29.

Lartillot, O., Toiviainen, P., and Eerola, T. (2008). A matlab toolbox for music information retrieval. In *Data analysis, machine learning and applications*, pages 261–268. Springer.

Lefebvre, A., Lecroq, T., and Alexandre, J. (2002). Drastic improvements over repeats found with a factor oracle. In Billington, E., Donovan, D., and Khodkar, A., editors, *Proceedings of the 13th Australasian Workshop on Combinatorial Algorithms*, pages 253–265.

Lévy, B., Bloch, G., and Assayag, G. (2012). Omaxist dialectics: Capturing, vizualizing and expanding improvisations. In *Proceedings of NIME 2012, Ann Arbor*.

Parncutt, R. (1994). A perceptual model of pulse salience and metrical accent in musical rhythms. *Music Perception*, 11(4):409–464.

Tillmann, B., Bigand, E., and Bharucha, J. (2000). Implicit learning of tonality: A self-organizing approach. *Psychological Review*, 107:885–913.

Toiviainen, P. and Krumhansl, C. (2003). Measuring and modeling real-time responses to music: The dynamics of tonality induction. *Perception*, 32:741–766.

Toiviainen, P., Tervaniemi, M., Louhivuori, J., Saher, M., Huotilainen, M., and Näätänen, R. (1998). Timbre similarity: Convergence of neural, behavioral, and computational approaches. *Music Perception: And Interdisciplinary Journal*, 16(2):223–241.

Vesanto, J. (1999). Self-organizing map in matlab: the som toolbox. In *Proceedings of the Matlab DSP Conference 1999*, pages 35–40, Espoo, Finland.