A Gentle Introduction to Somax

Somax is an interactive system which improvises around a musical material, aiming to provide a stylistically coherent improvisation while in real-time listening to and adapting to input from a musician. The system is trained on some musical material selected by the user, from which it constructs a corpus that will serve as a basis for the improvisation. The corpus and the output of the system is currently based on MIDI, but it is able to listen and adapt to both audio and MIDI input from a musician.

The main idea is that Somax should serve as a co-creative agent in the improvisational process, where the system after some initial tuning is able to listen and adapt to the musician in a self-sufficient manner. Of course, the input doesn't have to come from a live musician; any type of audio and/or MIDI input works, be it from an audio file, score editor, synthesizer or DAW. Somax also allows detailed parametric controls of its output and can even be played as an instrument in its own right. Also, the system isn't necessarily limited to a single agent or a single input source - it is possible to create an entire ensemble of agents where the user in detail can control how the agents listen to various input sources as well as to each other.

The goal of this text is to provide a brief introduction to Somax and provide the reader the with fundamental knowledge about how its interaction model works, which in turn should serve as a basis for making informed choices when tuning and interacting with the system. For a hands-on introduction to Somax and its user interface, also see the interactive tutorials that can be found in the same folder as this document.

The Corpus and the Navigation Model

As previously mentioned, the Somax system generates its improvisation material based on an external set of musical material, the «corpus». This corpus can be constructed from one or multiple MIDI files freely chosen by the user. In contrast to many other generative approaches, the system does not construct a model that eventually is independent of the material that was used to train it. Rather, the model is constructed directly on top of the original data and provides a way to navigate through it in a non-linear manner. 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.

When presented with some MIDI material, the first step is to segment the musical stream into discrete units or «slices», which are vertical (polyphonic) segments of the original midi file and where the duration of a slice is the distance between two note onsets.[1] Each slice is analyzed and classified with regards to a number of musical

[1] In recorded corpora (or any type of non-quantized MIDI) it is rare for any two notes that are perceived as simultaneous to be exactly simultaneous. Since one goal of Somax is to be able to maintain and reproduce the original timings within slices as recorded, notes with almost simultaneous onsets will still be grouped together in a single slice but with their internal timing offset preserved.

features related to its harmony, individual pitches, dynamics, etc., and these features will serve as the main basis for constructing the navigation model. This process is in a way similar to that of concatenative synthesis, where an audio signal is segmented into meaningful units, analyzed and recombined with respect to the analysis, but in this case (at least for now) based on MIDI data.

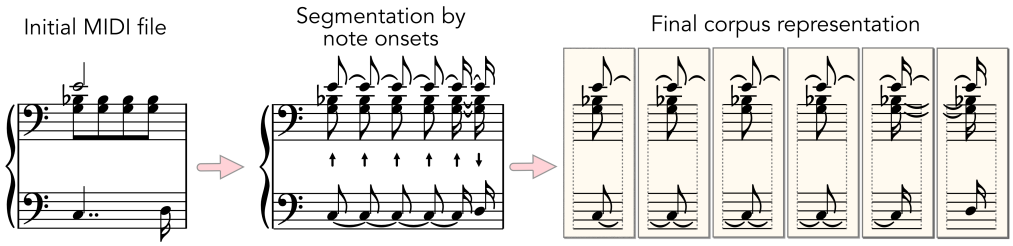

Figure 1: Constructing a corpus by segmenting midi data into «slices».

From the sequence of slices, the navigation model is constructed through the detection of common patterns within the musical material. Or more specifically, the procedure of detecting common patterns is repeated for each feature in the analysis, effectively resulting in a multilayer representation where each layer roughly corresponds to one feature, i.e., one layer for harmony, one for pitch, etc.

When a musician interacts with the system, a similar process of segmentation and multilayer analysis is computed on the input stream, and at each point in time the result of this process is matched to that of the navigation model, generating activations, or «peaks», at certain points in the corpus where the input corresponds to the model. Each peak corresponds to a point in the memory, so the entire set of peaks can effectively be seen linearly as a one-dimensional representation over the corpus' time axis (see figures 4 and 6 below for examples). The peaks in each layer are merged and scaled all according to how the system has been tuned, and finally the output slice is selected based on the distribution of the peaks.

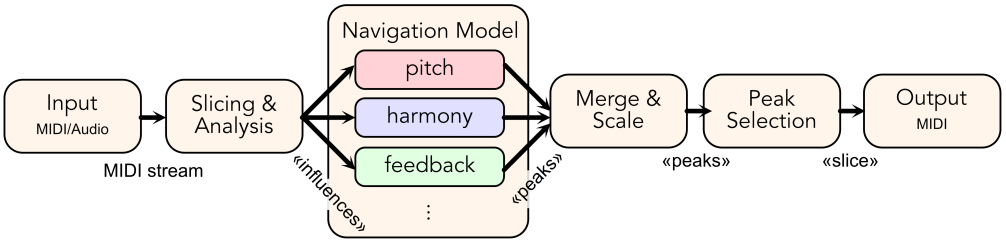

Figure 2: An overview of the steps through which the system generates its output at each given point in time.

The generated output of this process is an co-improvisation that recombines existing material in a way that's coherent with the sequential logic and statistical properties of the original material while at the same time adapting in real-time to match

the input from the musician. One benefit of this procedure compared to many other approaches of statistical machine learning is that the system can generate long improvisations from a small musical corpus, allowing the user detailed control over the style of the output or even specifically compose pieces of material intended to serve as a Somax corpora. Of course, it has to be recognized that this method is also one of the downsides with the model. The process of improvisation is to some extent reduced to a smart cut and pasting of pre-existing material, which is a very simplistic modelling of what improvisation is as a human skill. Still, the output is through the process of attempting to balance the internal logic of the corpus with the external logic of the musician often providing a mix of coherency and unexpectedness in a way that convincingly gives the impression of an active agent in the improvisational process.

Interacting with Somax

When interacting with Somax, there are three main concepts that are important to understand: «slices», «influences» and «peaks». A slice, as previously mentioned, is a short segment of the corpus and serves as the smallest building block of the output of the system. The slice can be manipulated to some extent (transposed, filtered with regards to voices/channels, etc.) but will always maintain most fundamental properties of the original corpus.

An «influence» is in a way conceptually very similar to a slice, but with a vastly different purpose. When Somax listens to a musician, this musical stream is segmented and analyzed with respect to its musical parameters similarly to how the corpus was constructed, but with a slightly different set of methods to be able to operate in real-time. The result of this process are discrete chunks of multilayer data or «influences», which the system uses to be able to compare the input to the corpus, where the main purpose of the influence is to act as the guide that determines the output of the system. The concept of an influence may initially seem like an implementation detail, but will become increasingly important for more complex configurations with multiple agents and/or multiple input sources. The main takeaway is that the system cannot listen directly to a musical input stream, but will need to translate it into influences, and that the process of tuning the listener can be a very important factor for the quality of the co-improvisation.

Finally, a «peak» is, again, a point in the corpus where the input corresponds to the model, or simply a match between an incoming influence and a corresponding slice that would serve as an output candidate. Each peak has a height, corresponding to a probability (or viability) of that particular slice as an output candidate. Unlike influences (which are visible in the interface) and slices (which are correlated to the audible output of the system), peaks are never interacted with directly, they're only part of the internal state of the system, but perhaps the most vital part. Each peak effectively corresponds to a slice in the corpus that could serve as an output at the current point in time, given the latest influence. Having a reasonable number of peaks is thus vital for the quality of the output, since having no peaks means that the output has not taken the musician's influences into account, and on the opposite side, in most cases a large number of peaks indicate that the matching is imprecise.

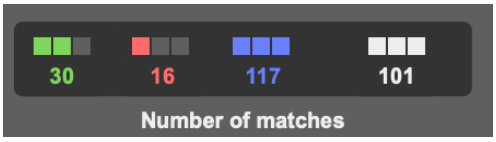

Figure 3: While the user doesn't interact directly with the peaks, they are still indicated in the user interface. Here, the colors green, red, blue and white correspond to the number of peaks in the feedback layer (more on this later), pitch layer, harmonic layer and total number of peaks after merge respectively.

To put the concept of peaks in context, let's dive a bit deeper in how the system works. While the musician is playing, Somax is at each detected onset segmenting the input into influences, carrying information about the pitch, harmony, etc. of what the musician currently is playing. This process is carried out by agents of the system called «influencers». This information is routed to a «player», which handles the entire process of matching and generating output. The influence is classified in multiple layers by the player, as briefly mentioned, where each layer corresponds to one musical dimension (e.g. harmony, melody) of the influence. In each layer, a model of the corpus with respect to the particular layer's musical dimension exists, and upon receiving an influence, the model will look for sequences in the corpus that match the sequence of most recent influences from the input, and in each of those places generate peaks.

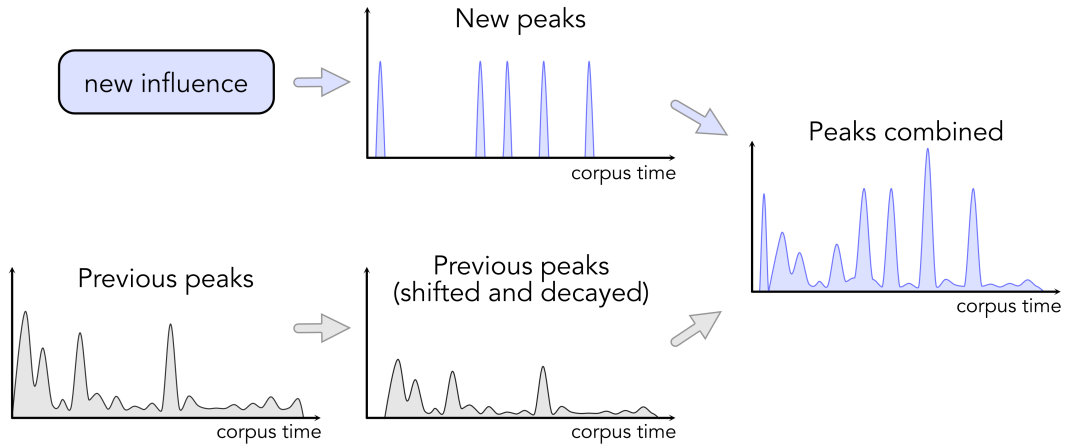

Figure 4: The process of shifting and decaying previous peaks in a single layer upon receiving new influences (the process of matching the incoming influence to the corpus has been omitted for clarity).

The system is also simulating a type of short-term memory inside this model by not immediately discarding peaks from previous influences, but rather shifting them along the time axis of the corpus and decaying their height corresponding to the amount of time that has passed, followed by merging them with the new set of peaks. This means that sequences continuously matching several consecutive influences will be more highly prioritized over others, as is illustrated in figure 4. Finally, the peaks from all layers will be merged together into a single set of peaks which the system

will use to probabilistically determine which slice is the best output candidate[2]. The result of this multilayer peak merging process is an output that will not just strictly match the harmony and pitch of the influence but rather improvise around the most recent history of influences with regards to the corpus, often (depending on how the parameters tuned) selecting peaks matching both the harmony and/or melody of the input but with an ability and agency to act more freely with regards to its history. In addition, there's also a layer which listens to the output of the system, a feedback layer, that can be used to balance the player's consistency with the input with its continuity with its own performance. The balance between the different layers as well as control over the decay time of old peaks, length of sequences to match in the memories, etc. are all available in the user interface as displayed in figure 5.

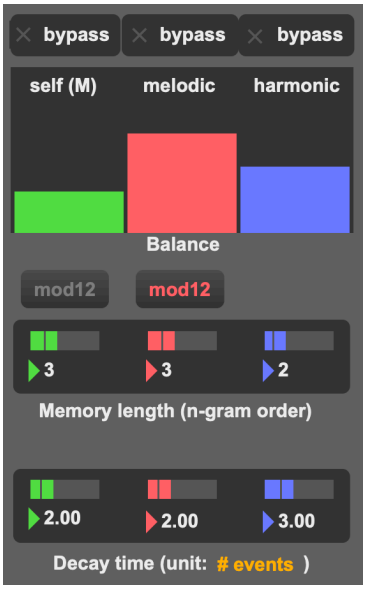

Figure 5: User interface to control the balance between the dimensions, length of matching sequences for each dimension as well as decay time of peaks.

If the concept of peaks isn't perfectly clear to you after reading this – don't worry! Go to the tutorial and start experimenting with the system while keeping one thing in mind: if the number of peaks is continuously zero or continuously too high[3], this is likely an indicator that the system is working poorly and should be retuned. If not – you're probably doing quite well.

Another important aspect of the interaction with Somax is its relation to time. According to the user's preference, each player can be assigned to either operate continuously in time as an autonomous agent, maintaining the pulse and exact within-slice timings of the original corpus (while possibly adapting to the tempo and/or phase of the input), or operate reactively, generating output synchronously as requested by the input. In the continuous case, this means that the player improvises freely over

[2]actually, in addition to this, there are a number of parameters that scale the height of the peaks individually with regard to a number of other musical parameters of choice, but this is thoroughly documented in the help files in max and will not be discussed here

[3]exactly how large "too high" is varies with the type of layer and context, but larger than 10% of the total number of slices in the corpus with no transpositions active could serve as a reference of "too high" that is valid for most layers and contexts

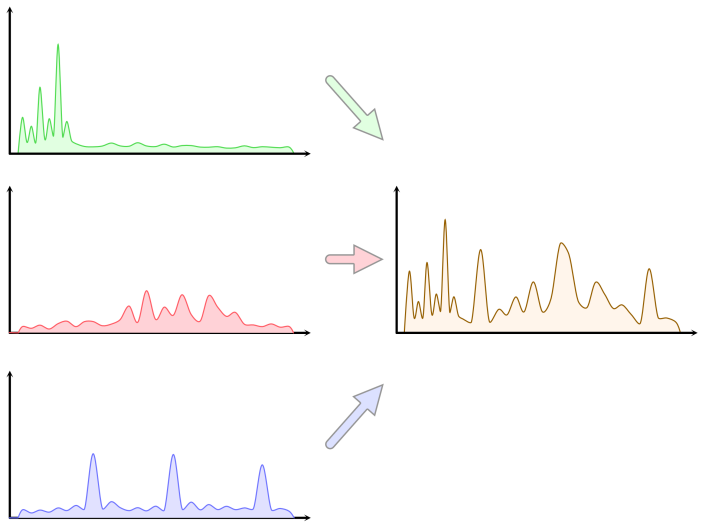

Figure 6: Three layers of peaks corresponding to different musical dimensions such as pitch, harmony, feedback, etc., being merged into one set of peaks before the final scaling and peak selection. Here, all three layers are weighted equally, but it is possible to balance the contribution from each of the layers.

time while still taking the influences of the musician into account, while in the reactive case, it synchronizes strictly (note-by-note) with the input. Of course, the player is in the latter mode not strictly limited to the input from where it receives its influences, but could be connected to a third source of some sort, for example any type of step-sequencer or other generative approaches, thus giving the user multiple options for controlling the temporal domain of the system.

Constructing the Corpus

By default, constructing a corpus is as simple as dragging a MIDI file to the «Corpus Builder» window in the user interface, from which Somax will build an internal representation with slices, as explained, along with annotations attached to relevant dimensions. There are however a number of things to consider here. If the input to a player and its internal corpus were made from similar source of materials, an ideal response of the player would tend to simply replicate the input. A great deal of variations or new musical situations will however arise from the discrepancies between the input and the corpus, and from the different mappings that the user can set for defining the musical dimensions considered by the player. As we are mostly talking about MIDI content here (players and MidiInfluencers) this is nothing else than a mapping between the MIDI channels and the melodic and harmonic dimensions.

This mapping occurs three-fold. Firstly, a player will have to know what subset (what MIDI channels) of its content (its corpus) maps to its internal melodic or harmonic dimensions, also called its «listening dimensions». Secondly, a source of influence (from an external MIDI input - a «MidiInfluencer» - or from the output of another player) will have to know how to map parts of its MIDI content to the influ-

ence's melodic and harmonic dimensions (these influence dimensions will be matched with the receptive player's «listening» dimensions). Finally the player must decide what part of its content is to be effectively played. The user will be able to control these three mappings in order to set precise interaction schemes.

For example, when creating a corpus from a polyphonic MIDI file (e.g. a string quartet with channels one to four assigned to the different instruments), the user could want the system to map notes from channel one (e.g. the lead violin) to the player's melodic listening dimension and notes from channels two to four (e.g. the remaining instruments) to its harmonic listening dimension. When a player, loaded with the said corpus, reacts to an incoming influence, it will continuously try to match the melodic and harmonic dimensions coming from the influence to is listening melodic and harmonic dimensions as mapped from the corpus. So, for example, if this player specifies channels two to four as output channels and listens to the melodic influence of a monophonic input from a musician, this means that the system will attempt to find slices where the content of channel one (e.g. the lead violin) corresponds to the input of the musician, while only outputting the content of channel two to four (e.g. the rest of the quartet). In this case, it would effectively generate an accompaniment to the input of the musician.

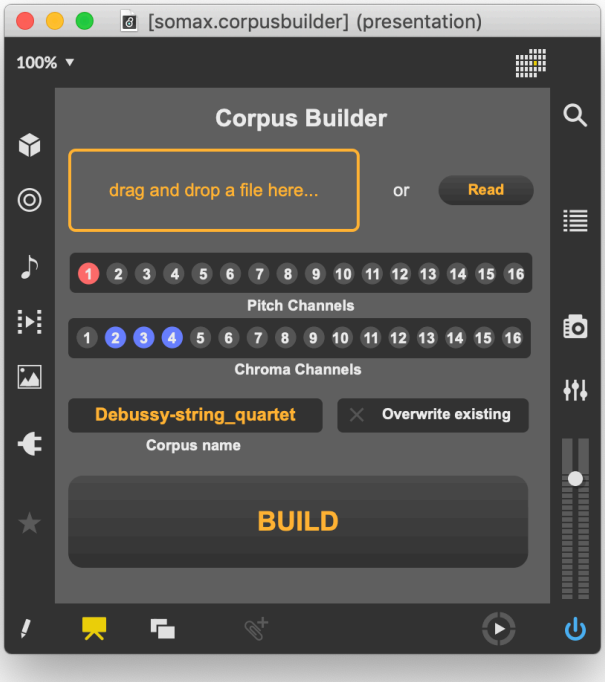

Figure 7: Specifying the melodic (pitch) and harmonic (chroma) dimensions when building the corpus that will feed the players.

For a more complex input to the system (e.g. multiple musicians, a multichannel MIDI file, etc.) it is in the same manner possible to set the mapping between the input's MIDI channels and the melodic and harmonic dimensions used to generate in-

fluences to the system. This can be very useful in systems with multiple players that are listening to and influencing each other. Returning to the example with the string quartet corpus, we could create a system with two players: player *P1*, generating its improvisation from the melodic part of the corpus (e.g. the lead violin's part) and another player *P2*, generating its improvisation from the harmonic part of the same corpus (e.g. the rest of the quartet). We could now have *P1* reactive to the melodic influence from a live musician; *P2* could react to *P1* by generating a harmonic texture coherent with *P1*'s directions; *P2* could also influence *P1*'s progression choices on a harmonic basis, thus competing with the external musician's influence.

Figure 8: Interconnecting the players to an audioinfluencer and one to another, with feedback from $P2$ to $P1$.

As shown in the above example, it is possible to create highly sophisticated networks of players influencing each other with detailed control over each player's listening and output dimensions. However, In simple cases (e.g. a single player with a corpus constructed from a piano MIDI file consisting of one or two channels) it is not necessary to focus on this particular aspect of the system - the default settings[4] will suffice. While the "ideal" output in this case, as mentioned in the beginning of this section, would be an identical replication of the input, the discrepancies between the input and the corpus will almost always ensure that the output is much more dynamic than a simple replication. Still, being aware of these intricacies will, once you've familiarized yourself with Somax, be very helpful for configuring the system for specific situations and improving the quality of the output.

[4]By default, all channels are mapped to all dimensions in all players and influencers. In this case, the whole musical content will influence both the harmonic dimension as well as the melodic (by default, the highest note registered will feed the melodic dimension although you can specify otherwise, e.g. bass line, etc.)

A Few Concluding Words

Hopefully, you will by now have an idea of what Somax is, what it does and even some understanding of how it does it. Some of these concepts may be a bit vague at the moment, but if you're reading this for the first time you'll certainly know enough to get started with the interactive tutorials and some initial interaction with Somax. Inside of the package, you will find the tutorial («tutorial.maxpat»), which is based on a condensed version of Somax with a single player connected to an audio influencer and a midi influencer. You will also find the full version of Somax («somax2.maxpat»), which allows dynamic creation of players and influencers as well as dynamic routing between players and/or influencers. Each of the main Somax modules have a help file (accessible by pressing the «?»-button) which in detail explains how to use the model, its parametric controls as well as outlines a number of use cases relevant for that particular module.

If you at a later point want to experiment with more advanced configurations, it might be useful to return to this text after you've acquired some hands-on experience with the system, but depending on how you want to use the system, your current knowledge may be sufficient. If you, on the other hand, want to know even more about the system and its implementation, you can find a number of articles, technical reports and other resources at http://repmus.ircam.fr/somax/home.

Annex: Documentation & Tutorials Examples[5]

Figure 9: Help files are available for all modules and can be accessed by pressing the «?»-mark in the corresponding Somax module.

[5]Disclaimer: These pictures might not exactly reflect the current state of the UI - please refer to the actual documentation inside Max for latest version.

Figure 10: Each help file contains a number of tabs outlining a particular aspect of relevance for the module.

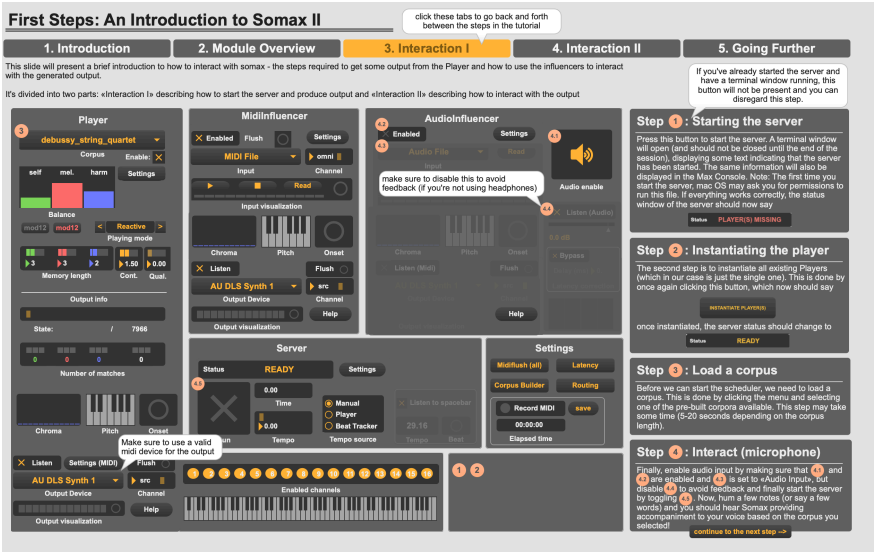

Figure 11: The tutorial describes how to properly initialize and use the model, step-by-step.