The Somax 2 Theoretical Model

Rev. 0.2.0

Joakim Borg

April 16, 2021

Somax 2 is a new version of the Somax reactive co-improvisation paradigm by. G. Assayag, with a completely redesigned architecture and UI written by Joakim Borg.

Legacy Somax versions by Laurent Bonnasse-Gahot and Axel Chemla-Romeu-Santos.

The Somax 2 project is part of the ANR project MERCI (Mixed Musical Reality with Creative Instruments) and the ERC project REACH (Raising Co-creativity in Cyber-Human Musicianship).

PI : Gérard Assayag
Music Representation Team
IRCAM STMS Lab (CNRS, Sorbonne University, Ministry of Culture).

`repmus.ircam.fr/impro`

Contents

Chapter 1

Overview

This report describes the theoretical model of the Somax 2 environment, which serves as a basis for the implementation of the system as described in [1]. 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 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.

This system, of which earlier versions have been presented in [2] and [3], generates musical material based on an external, already existing material. The principle through which the system generates new material is similar to that of concatenative synthesis, which in turn is to some extent alike that of granular synthesis, where very short pieces of audio are sampled from a preselected material and recombined. In this system, however, the grains (or ≪slices≫, as they will be denoted from here on) are much longer than in a granular synthesizer - either the duration of a beat or the length between two note onsets - and the output selection is based on listening to a second, external input, and the system is continuously selecting the most suitable slice using statistical machine learning. In other words, the system is improvising around a musical material and in real-time adapting to a musician (or any other sound source).

In practice, this behaviour is realized by learning the sampled material, which may be either audio or MIDI, in multiple layers, where each layer listens to a single feature (or ≪trait≫, as they will be denoted from here on) of the material, for example pitch, chroma, MFCC:s, velocity, etc. At runtime, each layer then listens to the corresponding trait of a second external audio and/or MIDI source and continuously matches this to the learned material, generating activations in the memory where matches are found. The output is then selected from the point in the memory with the most activity, after the activities in all layers has been merged and scaled. The activities generated at earlier points in time also remain in the memory for some time, thus impacting future time steps and with that simulating a short-term memory with respect to the original material.

The process of learning the sampled material or ≪constructing a corpus≫ will be described in section 2. The listening, from here on denoted as ≪influencing≫, is described in section 3, and the generation of output, which for practical reasons is decoupled from the influencing process, is described in section 4. Section 5 introduces the concept of transforms and its implications for the Somax model, and finally section 6 introduces a number of user-controller parametric filters to allow further control of the output.

Chapter 2

Corpus

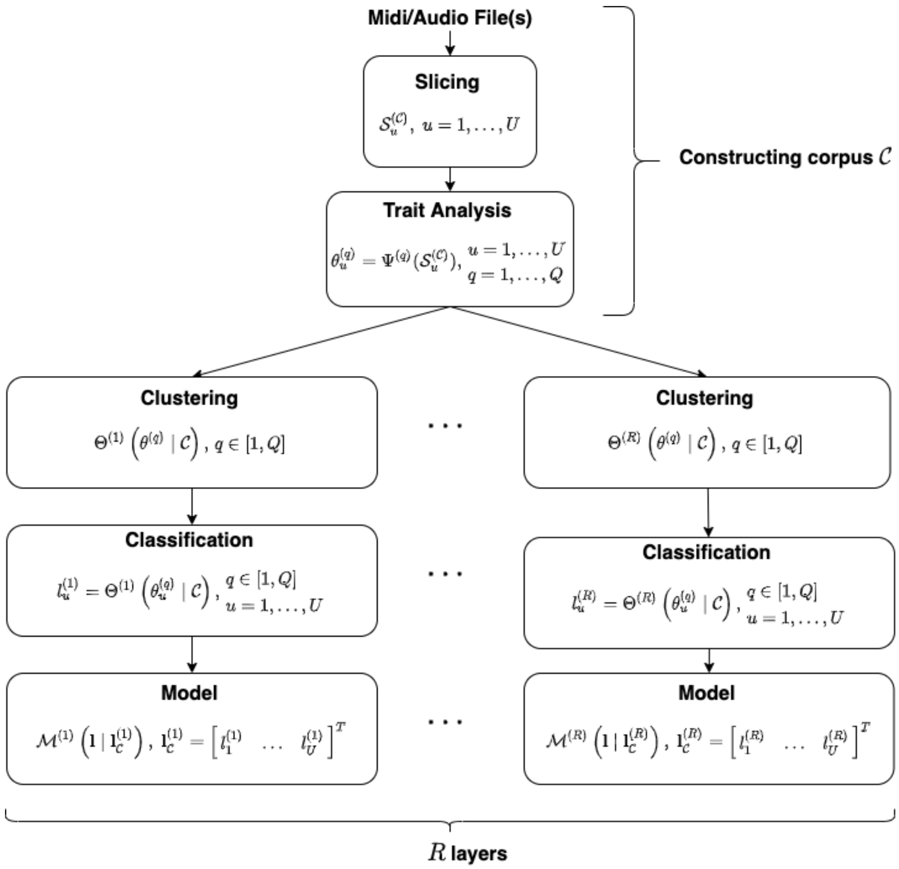
Midi/Audio File(s)
Slicing
$\mathcal{S}_u^{(\mathcal{C})},\ u = 1,\ldots,U$
Constructing corpus $\mathcal{C}$
Trait Analysis
$\theta_u^{(q)} = \Psi^{(q)}(\mathcal{S}_u^{(\mathcal{C})}),\ u = 1,\ldots,U,\ q = 1,\ldots,Q$
Clustering
$\Theta^{(1)}\left(\theta^{(q)} \mid \mathcal{C}\right),\ q \in [1,Q]$
Clustering
$\Theta^{(R)}\left(\theta^{(q)} \mid \mathcal{C}\right),\ q \in [1,Q]$
Classification
$l_u^{(1)} = \Theta^{(1)}\left(\theta_u^{(q)} \mid \mathcal{C}\right),\ q \in [1,Q],\ u = 1,\ldots,U$
Classification
$l_u^{(R)} = \Theta^{(R)}\left(\theta_u^{(q)} \mid \mathcal{C}\right),\ q \in [1,Q],\ u = 1,\ldots,U$
Model
$\mathcal{M}^{(1)}\left(\mathbf{l} \mid \mathbf{l}_{\mathcal{C}}^{(1)}\right),\ \mathbf{l}_{\mathcal{C}}^{(1)} = \begin{bmatrix} l_1^{(1)} & \ldots & l_U^{(1)} \end{bmatrix}^T$
Model
$\mathcal{M}^{(R)}\left(\mathbf{l} \mid \mathbf{l}_{\mathcal{C}}^{(R)}\right),\ \mathbf{l}_{\mathcal{C}}^{(R)} = \begin{bmatrix} l_1^{(R)} & \ldots & l_U^{(R)} \end{bmatrix}^T$
R layers

Figure 2.1: The main steps in constructing and modelling a Corpus.

The corpus $\mathcal{C}$ is the basis of Somax from which all output material will be drawn. It is constructed from one or multiple audio and/or midi files, which are segmented into short slices $\mathcal{S}$, where each segment is analyzed with respect to a number of traits θ, clustered and classified in multiple layers - each layer with respect to a single trait - and finally modelled according to a specific data model. This procedure can be seen in figure 2.1, and each step will be described in detail in the following sections.

2.1 Slicing

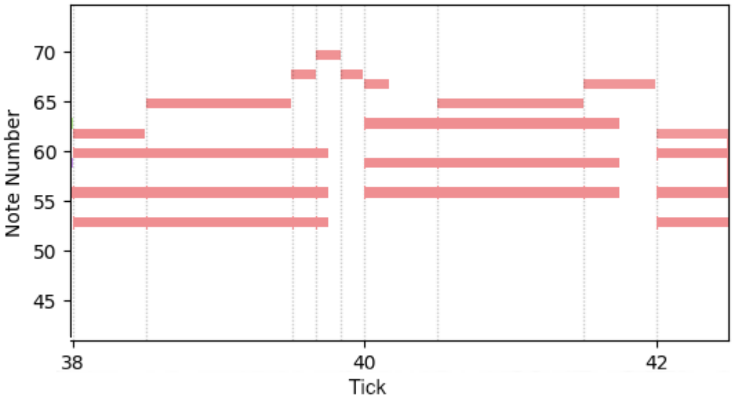

Figure 2.2: Example of slicing a short segment of a midi file, where the slices are represented by dashed vertical lines and notes represented by red horizontal bars.

The first step in constructing and modelling the corpus is to parse the midi/audio files and segment the content into slices along the time axis. The start of each slice is for midi file determined by each midi notes onset (see figure 2.2 and for audio files determined by each beat, which is estimated by either the dynamic programming algorithm [4] for audio files with a fixed tempo or using predominant local pulse estimation [5] for audio files with a dynamic tempo.

Each slice $\mathcal{S}^{(\mathcal{C})}$ is assigned an index u, an onset time $t^{(\mathcal{C})}$ (in ticks, where 1 tick correspond to one beat), a duration d (in ticks), a tempo ζ (in BPM) determined either by the midi tempo at that specific point in time or estimated from the inter-onset interval between two beats for audio files. The slice is also assigned an absolute onset time τ_u and an absolute duration δ_u, both in milliseconds, that will be relevant when determining the output position in audio files. Each slice is also analyzed with respect to a number of traits $\theta^{(q)}$, $q = 1, \dots, Q$ which will be described in section 2.2. More formally, we have a corpus of length U defined as

$$\mathcal{C} = \left\{ S_1^{(\mathcal{C})}, S_2^{(\mathcal{C})}, \dots, S_U^{(\mathcal{C})} \right\} \tag{2.1}$$

where

$$\mathcal{S}_u^{(\mathcal{C})} = \left\{ t_u^{(\mathcal{C})}, d_u, \zeta_u, \tau_u, \delta_u, \theta_u^{(1)}, \dots \theta_u^{(Q)} \right\}, \quad u \in [1, U]. \tag{2.2}$$

2.2 Trait Analysis

Note that all parameters in equation 2.2 apart from $\theta^{(1)}, \ldots, \theta^{(Q)}$ are only related to timing, so the purpose of the traits θ is to store any other feature of the slice, for example pitch, velocity, harmonic content, etc. These traits will later be used for clustering, classification and modelling the corpus. Another purpose of modelling each slice in terms of traits is to make the model format-agnostic, i.e. independent of whether the corpus is based on midi or audio data. Formally, the procedure of calculating trait $\theta_u^{(q)}$ of slice $\mathcal{S}_u^{(\mathcal{C})}$ can be described as

$$\theta_u^{(q)} = \Psi^{(q)} \left(\mathcal{S}_u^{(\mathcal{C})}, \left[\mathcal{S}_{u-1}^{(\mathcal{C})}, S_{u-2}^{(\mathcal{C})}, \ldots \right] \right), \quad u = 1, \ldots, U \tag{2.3}$$

where $\Psi^{(q)}$ denotes a function depending on $\mathcal{S}_u^{(\mathcal{C})}$ (and optionally previous slices) for calculating trait $\theta_u^{(q)}$. Once the trait analysis is completed, the process of constructing (but not modelling) the corpus is completed.

2.3 Clustering and Classification Modelling

The completed corpus $\mathcal{C}$ is modelled in R layers, where each layer $r \in [1, R]$ has its own clustering $\Theta^{(r)}$ and model $\mathcal{M}^{(r)}$. The clustering in layer r will be described as

$$\Theta^{(r)} \left(\theta^{(q)} \mid \mathcal{C} \right), \quad q \in [1, Q], \tag{2.4}$$

which in other words mean that each layer's clustering is constructed with regards to a single trait $\theta^{(q)}$ given a corpus $\mathcal{C}$. In practice, however, not all clusterings rely on $\mathcal{C}$ - there are both *absolute* and *relative* clusterings, which are described further in section **??**. Once a clustering has been constructed in layer r, each slice $\mathcal{S}_u^{(\mathcal{C})}$, $u = 1, \ldots, U$ in the corpus will be classified with respect to its corresponding parameter $\theta_u^{(q)}$, i.e.

$$l_u^{(r)} = \Theta^{(r)} \left(\theta_u^{(q)} \mid \mathcal{C} \right), \quad u = 1, \ldots, U \tag{2.5}$$

where $l_u^{(r)} \in \mathbb{Z}$ denotes the label of slice $\mathcal{S}_u^{(\mathcal{C})}$ in layer r with respect to trait $\theta^{(q)}$.

Finally, a model $\mathcal{M}^{(r)}$ is constructed from the collected labels, i.e.

$$\mathcal{M}^{(r)} \left(\boldsymbol{l} \mid \boldsymbol{l}_{\mathcal{C}} \right), \quad \boldsymbol{l}_{\mathcal{C}} = \begin{bmatrix} l_1^{(r)} & \ldots & l_U^{(r)} \end{bmatrix}^T. \tag{2.6}$$

where $\mathcal{M}$ can be described as a mapping from a vector of labels $\boldsymbol{l}$ to a set of slices, i.e.

$$\mathcal{M} \colon \boldsymbol{l} \to \left\{ \mathcal{S}_{u_1}^{(\mathcal{C})}, \ldots, S_{u_j}^{(\mathcal{C})} \right\}, \quad u_1, \ldots, u_j \in [1, U]. \tag{2.7}$$

$\mathcal{M}$ can be described as a simplified, unweighted n-gram model which simply returns all slices that matches the given input. The process of constructing $\mathcal{M}$ is described in algorithm 1, where γ denotes the n-gram order, κ denotes the given input at each time step and $\mathcal{M}$ here is modelled as a map where $\mathcal{M}\left[\kappa\right]$ denotes the values at the map's index κ.

Algorithm 1 Constructing $\mathcal{M}$

```
for $u = \gamma$ to $U$ do
    $\kappa = [l_{u-\gamma}, \ldots, l_u]$
    if $\mathcal{M}[\kappa]$ exists then
        $\mathcal{M}[\kappa] := \mathcal{M}[\kappa] \cup \left\{ \mathcal{S}_u^{(\mathcal{C})} \right\}$
    else
        $\mathcal{M}[\kappa] := \left\{ \mathcal{S}_u^{(\mathcal{C})} \right\}$
    end if
end for
```

Chapter 3

Influence

The influence process takes a continuous stream $\mathcal{K}$ of midi/audio data, segments it into slices, analyzes each slice with regards to its traits and determining where the corpus (or more specifically, at which temporal positions the models $\mathcal{M}$ constructed from the corpus) matches the incoming stream. These positions will later be used to determine the most suitable output. An overview of the influence process can be seen in figure 3.1.

One key aspect of Somax is that the influence process has a short-time memory, so that influences generated at previous time steps in the stream maintain an impact on the output for a certain amount of time. How this behaviour is simulated will be described in section 3.4.

3.1 Slicing

The slicing procedure of the influence process is almost identical to the procedure described in section 2.1, with the exception that it in the case of a real-time stream uses different algorithms for onset detection and beat estimation (see [3] for details). More formally, we will define the continuous stream $\mathcal{K}$ with an (in most cases undefined) length V as

$$\mathcal{K} = \left\{ \mathcal{S}_1^{(\mathcal{K})}, \dots, \mathcal{S}_V^{(\mathcal{K})} \right\} \tag{3.1}$$

where

$$\mathcal{S}_v^{(\mathcal{K})} = \left\{ t_v^{(\mathcal{K})}, d_v, \zeta_v, \theta_v^{(1)}, \dots \theta_v^{(Q)} \right\}, \quad v \in [1, V]. \tag{3.2}$$

While the differences between equations 2.2 and 3.2 seem to be that of replacing one index for another, there is one key difference that will be important in later sections: the difference between corpus time, $t^{(\mathcal{C})}$, and continuous time $t^{(\mathcal{K})}$. The corpus time $t_u^{(\mathcal{C})}$, $u = 1, \dots, U$ denotes the position of slice $\mathcal{S}_u^{(\mathcal{C})}$ in the corpus which will always be fixed. The influence time $t_v^{(\mathcal{K})}$ denotes the time of the current state v in the process of generating and is closely related to scheduling, which is described in [1].

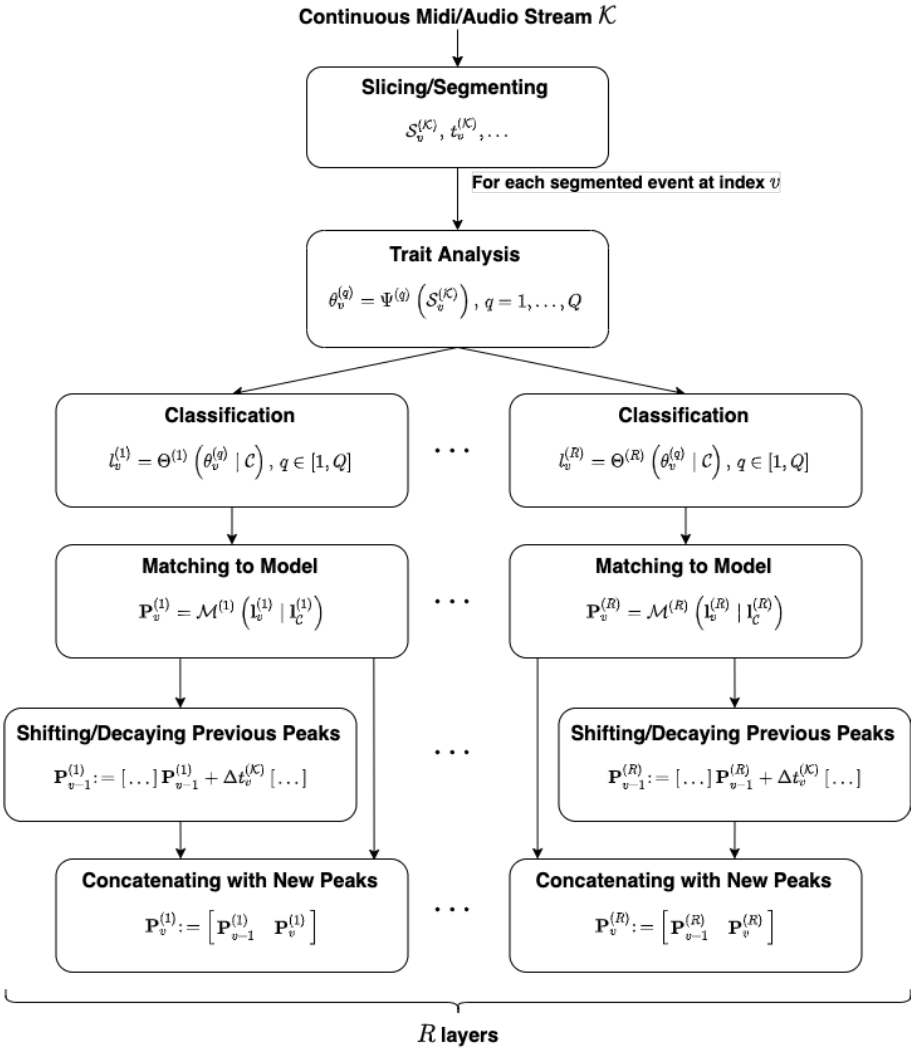
Continuous Midi/Audio Stream $\mathcal{K}$
Slicing/Segmenting
$\mathcal{S}_v^{(\mathcal{K})}, t_v^{(\mathcal{K})}, \dots$
For each segmented event at index v
Trait Analysis
$\theta_v^{(q)} = \Psi^{(q)}\left(\mathcal{S}_v^{(\mathcal{K})}\right), q = 1, \dots, Q$
Classification
$l_v^{(1)} = \Theta^{(1)}\left(\theta_v^{(q)} \mid \mathcal{C}\right), q \in [1, Q]$
. . .
Classification
$l_v^{(R)} = \Theta^{(R)}\left(\theta_v^{(q)} \mid \mathcal{C}\right), q \in [1, Q]$
Matching to Model
$\mathbf{P}_v^{(1)} = \mathcal{M}^{(1)}\left(\mathbf{l}_v^{(1)} \mid \mathbf{l}_\mathcal{C}^{(1)}\right)$
. . .
Matching to Model
$\mathbf{P}_v^{(R)} = \mathcal{M}^{(R)}\left(\mathbf{l}_v^{(R)} \mid \mathbf{l}_\mathcal{C}^{(R)}\right)$
Shifting/Decaying Previous Peaks
$\mathbf{P}_{v-1}^{(1)} := [\dots]\,\mathbf{P}_{v-1}^{(1)} + \Delta t_v^{(\mathcal{K})}\,[\dots]$
. . .
Shifting/Decaying Previous Peaks
$\mathbf{P}_{v-1}^{(R)} := [\dots]\,\mathbf{P}_{v-1}^{(R)} + \Delta t_v^{(\mathcal{K})}\,[\dots]$
Concatenating with New Peaks
$\mathbf{P}_v^{(1)} := \left[\mathbf{P}_{v-1}^{(1)} \quad \mathbf{P}_v^{(1)}\right]$
. . .
Concatenating with New Peaks
$\mathbf{P}_v^{(R)} := \left[\mathbf{P}_{v-1}^{(R)} \quad \mathbf{P}_v^{(R)}\right]$
R layers

Figure 3.1: The main steps in parsing a continuous midi/audio stream and influencing somax.

3.2 Trait Analysis, Classification and Matching to Model

Trait analysis and classification of a slice $\mathcal{S}_v^{(\mathcal{K})}$ is identical to the procedure described in section 2, more specifically to equation 2.3 and 2.5 respectively, with the only difference that it occurs in continuous time as soon as a slice $\mathcal{S}_v^{(\mathcal{K})}$ has been segmented. The system uses the R layers that were used model the corpus, and in each layer the clustering created from the corpus is used for classification. In other words, at time step $t_v^{(\mathcal{K})}$ corresponding to influence slice $\mathcal{S}_v^{(\mathcal{K})}$ with traits $\theta_v^{(1)}, \dots, \theta_v^{(Q)}$, we get one label $l_v^{(r)}$ per layer $r = 1, \dots, R$ so that

$$l_v^{(r)} = \Theta^{(r)} \left(\theta_v^{(q)} \mid \mathcal{C} \right) \tag{3.3}$$

where $\theta_v^{(q)}$ denotes the single trait used in layer r at the time step corresponding to index v.

3.3 Matching Slice to Model

In each layer $r = 1, \dots R$, at time step $t_v^{(\mathcal{K})}$ corresponding to influence slice $\mathcal{S}_v^{(\mathcal{K})}$, a vector will be constructed from the previous $k \in \mathbb{Z}_*$ labels, i.e.

$$\boldsymbol{l}_v^{(r)} = \begin{bmatrix} l_{v-k}^{(r)} & \dots & l_{v-1}^{(r)} & l_v^{(r)} \end{bmatrix}, \tag{3.4}$$

and in accordance with the definition in equation 2.7 generate a set of slices $\Sigma_v^{(r)} = \left\{ \mathcal{S}_{u_1}^{(\mathcal{C})}, \dots, S_{u_j}^{(\mathcal{C})} \right\}, \quad u_1, \dots, u_j \in [1, U]$. From $\Sigma_v^{(r)}$, a matrix of peaks $\boldsymbol{P}_v^{(r)} \in \mathbb{R}^{2 \times j}$ is constructed so that

$$\boldsymbol{P}_v^{(r)} = \begin{bmatrix} \boldsymbol{p}_{u_1} & \dots & \boldsymbol{p}_{u_j} \end{bmatrix} \tag{3.5}$$

where

$$\boldsymbol{p}_{u_m} = \begin{bmatrix} t_{u_m}^{(\mathcal{C})} \\ y_{u_m} \end{bmatrix}, \qquad m = 1, \dots, j \tag{3.6}$$

where y_{u_m} is a score (height) designated to each peak by the model and $t_{u_m}^{(\mathcal{C})}$ is the temporal position of the corresponding slice's onset.

3.4 Peaks

The final step in the influence process is to in each layer $r = 1, \dots R$ add the peaks $\boldsymbol{P}_v^{(r)}$ generated at time step $t_v^{(\mathcal{K})}$ corresponding to index v to the previous set of peaks $\boldsymbol{P}_{v-1}^{(r)}$ generated at the previous time step $t_{v-1}^{(\mathcal{K})}$. The purpose of this behaviour is, as was described in section 3, to simulate a short-term memory, where previous influences maintain a degree of impact on the output.

To achieve this, the positions of the previous peaks are shifted (in *corpus* time $t^{(\mathcal{C})}$) corresponding to the influence time $t^{(\mathcal{K})}$ elapsed since the previous influence, and the scores are decayed exponentially corresponding to the same interval scaled by a factor $\tau \in \mathbb{R}$, i.e.

$$\boldsymbol{P}_{v-1}^{(r)} := \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\Delta t_v^{(\mathcal{K})} / \tau \right) \end{bmatrix} \boldsymbol{P}_{v-1}^{(r)} + \Delta t_v^{(\mathcal{K})} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 0 & 0 & \dots & 0 \end{bmatrix} \tag{3.7}$$

where

$$\Delta t_v^{(\mathcal{K})} = t_v^{(\mathcal{K})} - t_{v-1}^{(\mathcal{K})}. \tag{3.8}$$

Finally, the peak matrix $\boldsymbol{P}_v^{(r)}$ is updated by concatenating it with the shifted and decayed peaks from the previous time step, i.e.

$$\boldsymbol{P}_v^{(r)} := \begin{bmatrix} \boldsymbol{P}_{v-1}^{(r)} & \boldsymbol{P}_v^{(r)} \end{bmatrix}. \tag{3.9}$$

Chapter 4

Generate

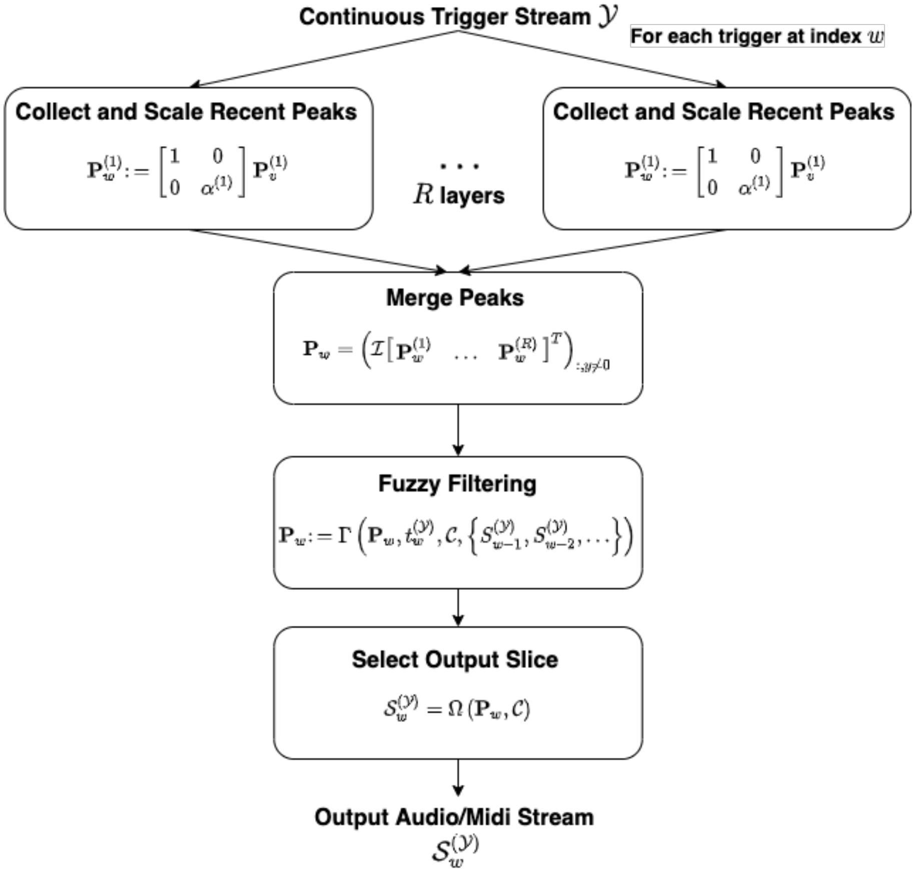
Continuous Trigger Stream $\mathcal{Y}$
For each trigger at index w
Collect and Scale Recent Peaks
$\mathbf{P}_w^{(1)} := \begin{bmatrix} 1 & 0 \\ 0 & \alpha^{(1)} \end{bmatrix} \mathbf{P}_v^{(1)}$
...
R layers
Collect and Scale Recent Peaks
$\mathbf{P}_w^{(1)} := \begin{bmatrix} 1 & 0 \\ 0 & \alpha^{(1)} \end{bmatrix} \mathbf{P}_v^{(1)}$
Merge Peaks
$\mathbf{P}_w = \left(\mathcal{I}\left[\mathbf{P}_w^{(1)} \quad \dots \quad \mathbf{P}_w^{(R)}\right]^T\right)_{:,y\not\angle 0}$
Fuzzy Filtering
$\mathbf{P}_w := \Gamma\left(\mathbf{P}_w, t_w^{(\mathcal{Y})}, \mathcal{C}, \left\{S_{w-1}^{(\mathcal{Y})}, S_{w-2}^{(\mathcal{Y})}, \dots\right\}\right)$
Select Output Slice
$\mathcal{S}_w^{(\mathcal{Y})} = \Omega\left(\mathbf{P}_w, \mathcal{C}\right)$
Output Audio/Midi Stream
$\mathcal{S}_w^{(\mathcal{Y})}$

Figure 4.1: The main steps in generating output from a continuous trigger stream.

So far, both section 2 and 3 have described a number of procedures for updating the internal state of the system without generating any output. This section will describe the steps taken to generate output from a continuous trigger stream $\mathcal{Y}$ of (an often undefined) length W based on the state set by recent influences, for which the procedure is outlined in figure 4.1

4.1 Triggers

While the concept of a continuous audio/midi stream is rather self-explanatory, the concept of a continuous trigger stream might need some explanation. Similarly to a continuous midi stream, a trigger stream is a stream of discrete events occurring at a time $t_w^{(\mathcal{Y})}$, but without any external information. In other words, the behaviour is that of a signal telling the system when to generate output.

In practice, the trigger stream is often correlated to the influencing audio/midi stream, either by midi note onsets, beat onsets or pitch detection onsets (for more details, see [3]) depending on the mode of the system, but may also be completely decorrelated to incoming influences. This behaviour depends on the mode of the system, which is discussed in [1]. In all cases, influencing and generating operate interleaved along the same time axis, so the behaviour of $t_w^{(\mathcal{Y})}$ is similar to the behaviour of $t_v^{(\mathcal{K})}$.

4.2 Collecting, Scaling and Merging Peaks

When a trigger is received, the first step is to collect and scale the peaks from all layers $r = 1, \ldots, R$. Each layer will have a designated weight $\alpha^{(r)}$ to scale the scores of each layer, i.e.

$$\boldsymbol{P}_w^{(r)} = \begin{bmatrix} 1 & 0 \\ 0 & \alpha^{(r)} \end{bmatrix} \boldsymbol{P}_v^{(r)}, \quad r = 1, \ldots, R \tag{4.1}$$

where $\boldsymbol{P}_v^{(r)}$ here denotes the peak matrix generated by the most recent influence step v.

The scaled peaks $\boldsymbol{P}_w^{(r)}$ from all layers $r = 1, \ldots R$ are then gathered into a single matrix where any peaks that occur sufficiently close to each other in corpus time $t^{(\mathcal{C})}$ are summed, i.e.

$$\boldsymbol{p}_i = \begin{bmatrix} t_i^{(\mathcal{C})} \\ y_i + y_j \end{bmatrix} \quad \text{if} \quad \left| t_i^{(\mathcal{C})} - t_j^{(\mathcal{C})} \right| < \varepsilon \quad \forall \boldsymbol{p}_i, \boldsymbol{p}_j \in \boldsymbol{P}_w, \tag{4.2}$$

for some interval ε, where

$$\boldsymbol{P}_w = \begin{bmatrix} \boldsymbol{P}_w^{(1)} & \ldots & \boldsymbol{P}_w^{(R)} \end{bmatrix}. \tag{4.3}$$

As the number of peaks n_w at the time at index w after v influences has a theoretical worst case of $n_w = \mathcal{O}(vRU)$, some optimization is required to solve equation 4.2, which by default has a time complexity of $\mathcal{O}\left(n_w^2\right)$. In practice, this is solved in linear time by multiplying the transposed peaks with a binary interpolation matrix $\mathcal{I} \in \mathbb{Z}_{[0,1]}^{m \times n}$

with $m = \lfloor 1/\varepsilon \rfloor$ rows and n_w columns, and selecting all non-zero columns from the transposed output, i.e.

$$\mathbf{\Pi}_w = \mathcal{I}\boldsymbol{P}_w^T \tag{4.4}$$

$$\boldsymbol{P}_w := \left(\mathbf{\Pi}_w^T\right)_{:\,,\,y\neq 0} \tag{4.5}$$

where

$$\left(\mathcal{I}\right)_{i,j} = \begin{cases} 1 & \text{if} \quad \left\lfloor \frac{t_j^{(\mathcal{C})}}{\varepsilon t_U^{(\mathcal{C})}} \right\rfloor = i \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

resulting in a single merged peak matrix $\boldsymbol{P}_w \in \mathbb{R}^{2\times \hat{n}_w}$, $\hat{n}_w \leq n_w$ for the time step at index w.

4.3 Scaling Peaks Again - Fuzzy Filtering

In section 4.2, all peaks in each layer were scaled uniformly by a weight. Once merged, the peaks are scaled again with respect to a set of more elaborate algorithms $\left\{\Gamma^{(1)}, \dots, \Gamma^{(J)}\right\}$, based on the time of influence, each peak's corresponding corpus data and/or previously output slices, i.e.

$$\boldsymbol{P}_w := \Gamma^{(j)}\left(\boldsymbol{P}_w, t_w^{(\mathcal{Y})}, \mathcal{C}, \left\{\mathcal{S}_{w-1}^{(\mathcal{Y})}, \mathcal{S}_{w-2}^{(\mathcal{Y})}, \dots\right\}\right) \quad \forall j = 1, \dots, J \tag{4.7}$$

where

$$\Gamma^{(j)}\colon \boldsymbol{P}_x \to \boldsymbol{P}_z, \quad \boldsymbol{P}_z \in \mathbb{R}^{m_x \times n_x}. \tag{4.8}$$

This behaviour can be seen as a sort of fuzzy filtering, as opposed to the binary matching provided by each classification/model-layer, to further emphasize or de-emphasize peaks with regards to parameters that may be difficult to classify in a meaningful manner.

4.4 Generating Output

Finally, the output for the time step at index w is selected from $\mathcal{C}$ as the slice closest to the peak $\boldsymbol{p}_i \in \boldsymbol{P}_w$ with the highest score y_i, i.e.

$$\hat{\boldsymbol{p}}_w = \begin{bmatrix}\hat{t}^{(\mathcal{C})} & \hat{y}\end{bmatrix}^T = \left\{\boldsymbol{p}_j \mid \forall \boldsymbol{p}_i \in \boldsymbol{P}_w \colon y_j \geq y_i\right\}, \tag{4.9a}$$

$$\mathcal{S}_w^{(\mathcal{Y})} = \min_{\mathcal{S}_u^{(\mathcal{C})} \in \mathcal{C}} \left|\hat{t}^{(\mathcal{C})} - t_u^{(\mathcal{C})}\right|. \tag{4.9b}$$

If multiple vectors $\boldsymbol{p}_j$ fulfil the condition in equation 4.9a, a single one will be selected randomly. If on the opposite no vectors fulfil this condition, i.e. if the peak matrix $\boldsymbol{P}_w$ is empty, the slice $\mathcal{S}_{u+1}^{(\mathcal{C})}$ following the previously output slice will be used for output, i.e. given

$$\mathcal{S}_{w-1}^{(\mathcal{Y})} = \mathcal{S}_u^{(\mathcal{C})} \tag{4.10}$$

for some $u \in [1, U]$, we get

$$S_w^{(\mathcal{Y})} = \mathcal{S}_{u+1}^{(\mathcal{C})}. \tag{4.11}$$

Chapter 5

Transforms

A fundamental problem with the model used in Somax is the fact that the quality of the output heavily relies on how well the influence (input) matches the corpus. This lies in the nature of its corpus-based approach - if the n-gram model is unable to find any slices matching a given series of influence labels, the output cannot be generated in relation to the given input. For example, if a corpus is using only the pitches of the D major scale while the influence is strictly limited to the pitches of the Db major scale, finding harmonic or melodic matches between the two will be impossible as they have no tones in common. What they do have in common is however intervallic relationships between the pitches, and if we could use this in the matching algorithm, the number of matches would increase without reducing the quality of the individual matches. For this purpose, the concept of transforms was introduced.

5.1 Definition

A transform $\mathcal{T}$ is a set of functions $\Lambda\colon \mathbb{V} \to \mathbb{V}$ for an arbitrary domain $\mathbb{V}$ that fulfil a number of conditions. More specifically, $\mathcal{T} = \{\Lambda_{\mathcal{S}}, \Lambda_{\theta 1}, \dots, \Lambda_{\theta q}\}$ where

(i) $\Lambda_{\mathcal{S}}$ is defined for any slice $\mathcal{S}_u^{\mathcal{C}}$, $u = 1, \dots U$ given a corpus $\mathcal{C}$ of length U so that the content of $\mathcal{S}_u^{(\mathcal{T})} = \Lambda_{\mathcal{S}}\left(\mathcal{S}_u^{(\mathcal{C})}\right)$ is defined and can be played in real-time,

(ii) Each function $\Lambda_{\theta q}$, $q = 1, \dots, Q$ is defined for one trait $\theta^{(q)}$,

(iii) Each function $\Lambda_{\mathbf{v}}$ has an inverse $\Lambda_{\mathbf{v}}^{-1}$ defined so that for any arbitrary element $\mathbf{v} \in \mathbb{V}$,

$$\Lambda_{\mathbf{v}}^{-1}\left(\Lambda_{\mathbf{v}}\left(\mathbf{v}\right)\right) = \Lambda_{\mathbf{v}}\left(\Lambda_{\mathbf{v}}^{-1}\left(\mathbf{v}\right)\right) = \mathbf{v}. \tag{5.1}$$

Given a transform $\mathcal{T}$ fulfilling property (i) and (ii), we can transform each slice $\mathcal{S}_u^{(\mathcal{C})}$ in the corpus and given a sequence of influences look for matches in both the original sequence $\left\{\theta_1^{(q}, \dots, \theta_U^{(q)}\right\}$ as well as the transformed sequence $\left\{\Lambda_{\theta q}\left(\theta_1^{(q)}\right), \dots, \Lambda_{\theta q}\left(\theta_U^{(q)}\right)\right\}$ as of property (ii), while ensuring that a slice $\mathcal{S}_u^{(\mathcal{T})}$ can be output (as of property (i)) and matches the influence. The third property exists for performance reasons and will be discussed in the following sections.

5.2 Transposition

Given the importance of the pitch domain in the Somax architecture, either in terms of melodic content (Top Note Classifier, Pitch Class Classifier; see [1] for implementation) or harmonic content (chroma; again see [1]), the main transform described in this section is the Transposition transform. The intuition behind this transform is simple - given our previous example with a corpus in D and influence in Db, we simply transpose our corpus to Db.

More formally, we define a transposition $\mathcal{T}^{(n)}$ of $n \in [-5, 6]$[1] semitones as the set

$$\mathcal{T}^{(n)} = \left\{\Lambda_{\mathcal{S}}^{(n)}, \Lambda_{\theta Pt}^{(n)}, \Lambda_{\theta c}^{(n)}\right\} \tag{5.2}$$

where

(i) The function $\Lambda_{\mathcal{S}}^{(n)}\left(\mathcal{S}_u^{(\mathcal{C})}\right)$ is for MIDI[2] corpora defined as the set of notes $\mathcal{N}_u^{(\mathcal{T}n)} = \left\{m_1^{(\mathcal{T}n)}, \dots, m_k^{(\mathcal{T}n)}\right\}$ (per definitions in section 3.4.2 of [**?**]) where $m_k^{(\mathcal{T}n)}$ is the k:th note in $\mathcal{N}_u^{(\mathcal{C})}$ with its pitch altered by n, i.e.

$$m_k^{(\mathcal{T}n)}.\texttt{pitch} = m_k.\texttt{pitch} + n, \quad \forall m_k \in \mathcal{S}_u^{(\mathcal{C})}. \tag{5.3}$$

(ii.1) The function $\Lambda_{\theta Pt}^{(n)}$ given a pitch $\theta^{(Pt)} \in \mathbb{Z}_{[0,127]}$ is defined as the pitch shifted by n, i.e.

$$\Lambda_{\theta Pt}^{(n)}\left(\theta^{(Pt)}\right) = \theta^{(Pt)} + n \tag{5.4}$$

(ii.2) The function $\Lambda_{\theta c}^{(n)}$ given a chroma vector $\boldsymbol{\theta}^{(c)} \in \mathbb{R}_{[0,\infty]}^{12}$ is defined as the chroma vector rotated by n, i.e.

$$\Lambda_{\theta c}^{(n)}\left(\theta^{(c)}\right) = \mathbf{M}^{(n)}\boldsymbol{\theta}^{(c)}, \tag{5.5}$$

where the rotation matrix $\boldsymbol{M}^{(n)} \in \mathbb{Z}_{[0,1]}^{12\times 12}$ is given by

$$\left(\mathbf{M}^{(n)}\right)_{i+1,j+1} = \begin{cases} 1 & \text{if } i = j + n \mod 12 \\ 0 & \text{otherwise} \end{cases}, \quad i, j = 0, \dots, 11. \tag{5.6}$$

(iii) Finally, the inverses are defined identically to equations 5.3-5.6 but with all n-terms substituted by $-n$.

5.3 Procedure

The concept of transforms is by no means a novelty in Somax. In fact, it was already present in an early beta version of Somax 2 released back in 2018. The main problem of this version was however a huge decrease in performance, increasing the latency of the system by up to several hundreds of milliseconds (on a 2018 MacBook Pro 13"

[1] The domain $[-5, 6]$ was chosen to allow transpositions to all 12 pitch classes while ensuring that the transposed content is as close to the original timbre of the input as possible.

[2] While audio corpora are not currently supported as of Somax version 2.3, the same behaviour could be implemented for audio corpora by pitch shifting the temporal interval of the original audio file by a value corresponding to the number of semitones n.

with a corpus of length $U < 10000$) when using multiple of transforms. This section will outline how the new implementation of transforms has altered the three main processes of the system: (1) constructing and modelling the corpus, (2) influencing and (3) generating output[3], which will be evaluated in terms of performance in section 5.4.

For constructing and modelling the corpus, the implementation of transforms does in fact not introduce any changes. In section 5.2, the procedure was described as a transposition of the entire corpus, which would mean clustering, classification and constructing N parallel models of the corpus given N transforms. The problem with this approach is that it would increase load time significantly and modifying parameters of the clustering/classification or modelling modules would become too computationally heavy to handle at runtime. This approach would also lead to a significant increase in runtime computation time as the size of the corpus would effectively be N times as large. Instead, we're utilizing property (iii) to inverse-transform the incoming influences $\theta^{(\mathcal{K})}$ for each transform before classification, i.e. that for a transform $\mathcal{T}^{(n)}$ and a slice $\mathcal{S}_u^{(\mathcal{C})}$ containing the trait $\theta_u^{(\mathcal{S})}$ we're utilizing the fact that if

$$\Lambda_\theta^{(n)}\left(\theta_u^{(\mathcal{S})}\right) = \theta^{(\mathcal{K})} \tag{5.7}$$

then

$$\left(\Lambda_\theta^{(n)}\right)^{-1}\left(\theta^{(\mathcal{K})}\right) = \theta_u^{(S)}. \tag{5.8}$$

Then for each inverse-transformed influence, we're computing the same steps as in section 3 for each of the n inverse-transformed influences, with the only new addition being that a peak $\boldsymbol{p}$ which previously was defined as

$$\boldsymbol{p} = \begin{bmatrix} t^{(\mathcal{C})} \\ y \end{bmatrix} \tag{5.9}$$

now contains additional information about which transform was applied,[4] i.e.7

$$\boldsymbol{p} = \begin{bmatrix} t^{(\mathcal{C})} \\ y \\ \mathcal{T}^{(n)} \end{bmatrix}. \tag{5.10}$$

Finally, for the process of merging peaks and generating output, the former process is identical to the description in section 2.4.2 in [**?**] except that the concatenated peak matrix $\boldsymbol{P}_w$ is split into n matrixes - one per transform - so that peaks corresponding to similar corpus time $t^{(\mathcal{C})}$ but different transforms are not merged with each other. The latter process, the generation of output, contains an additional step, that of transforming the selected output slice $\mathcal{S}_w^{(\mathcal{Y})}$ in accordance with its corresponding peak's applied transform, i.e. the final output slice $\mathcal{S}_w^{(\mathcal{T}n)}$ is defined as

$$\mathcal{S}_w^{(\mathcal{T}n)} = \Lambda_\mathcal{S}^{(n)}\left(\mathcal{S}_w^{(\mathcal{Y})}\right) \tag{5.11}$$

where $\Lambda_\mathcal{S}^{(n)}$ is the function corresponding to the transform of the selected peak $\hat{\boldsymbol{p}}_w$ in the case where such a peak exist. If no peak exists, the default output is the slice

[3] For a thorough description of each of these sections, refer to chapter 2 in [**?**].

[4] For practical reasons, this transform is denoted $\mathcal{T}$ here but in the actual implementation it is stored as an integer reference, which means that it indeed can be stored in a vector $\boldsymbol{p} \in \mathbb{R}^3$.

following the previously output slice with the same transform applied, i.e. given the previous output,

$$\mathcal{S}_{w-1}^{(\mathcal{T}n)} = \Lambda_{\mathcal{S}}^{(n)} \left(\mathcal{S}_{w-1}^{(\mathcal{Y})} \right) \quad \text{where} \quad \mathcal{S}_{w-1}^{(\mathcal{Y})} = \mathcal{S}_{u}^{(\mathcal{C})} \tag{5.12}$$

for some $u \in [1, U]$, we get

$$\mathcal{S}_{w}^{(\mathcal{T}n)} = \Lambda_{\mathcal{S}}^{(n)} \left(\mathcal{S}_{u+1}^{(\mathcal{C})} \right). \tag{5.13}$$

5.4 Runtime Performance

To evaluate the increase in computation time (i.e. runtime latency) caused by the addition of transforms, a sequence of integration tests were carried out. In these tests, the computational cost of the two main runtime processes, influence and generate, were evaluated separately on a corpus consisting of $U = 7966$ slices under four different conditions: no additional transform applied (formally one transform of 0 semitones), three additional transforms applied, six additional transforms applied and 11 additional transforms applied. The same corpus was used both for the model and the influence, i.e.

$$\mathcal{S}_{u}^{(\mathcal{C})} = \mathcal{S}_{w}^{(\mathcal{K})} \quad w = u = 1, \ldots, U, \tag{5.14}$$

and iterated over linearly 10 times, in total yielding 79660 influence-generate steps, but at each step in the iteration, $p = 10^0, 10^1, \ldots, 10^4$ additional peaks with random scores, times and transforms were inserted to evaluate how the system performs under different conditions. The entire process was evaluated on a MacBook Pro 13-inch 2018 with a 2.3 GHz Intel Core i5 processor running MacOS 10.15.

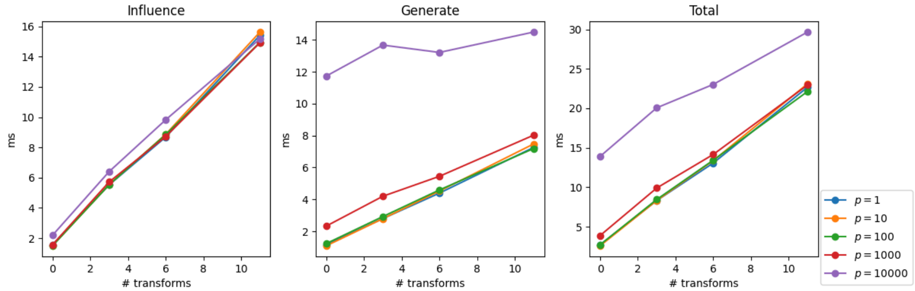

Figure 5.1: Average computational cost with respect to number of transforms.

As depicted in figure 5.1, the average computational cost increases linearly with the number of transforms and remains fairly constant independently of the number of added peaks.[5] For the generation process we also see a linear increase with the number of transforms, but the increase with relation to number of transforms is much less steep, while the computational cost is determined by the number of peaks to a

[5] The cost increases slightly with a large number of peaks, likely due to reallocation of memory being slightly more expensive in the final concatenation of peaks, but this cost does not have a significant impact on the overall result

much higher degree. In the third figure, we see the average cost of an entire influence-generation cycle, i.e. the time elapsed from when a performer presses a key to when the system responds. Without going too deep into psychoacoustics we can assume that most performers would consider a latency of 3.897 ms (0 additional transforms, $p = 1000$) as close to imperceivable, a latency of 9.911 ms (3 additional transforms, $p = 1000$) most likely acceptable, but above that, we run into issues of perceivable latencies, for example with 6 additional transforms at $p = 1000$ (14.16 ms), or even with 0 additional transforms at $p = 10000$ (13.91 ms).

The relation between computation time of the influence process and generate process are also important to take into account when further expanding the system. With the current relationship between estimated number of peaks and the number of classifiers, this implementation of transforms is ideal for runtime purposes, but should the system in the future move towards a higher degree of parallel classifiers, it's important to take this into account and possibly find other strategies for handling transforms in relation to classification.

Chapter 6

More Control Parameters

A number of new features and other architectural improvements have been added to the main runtime framework to improve the quality of the output as well as give the user a higher degree of parametric control. This section will briefly expand upon the notation presented in section 4 and then present each of the newly implemented features.

6.1 Some Additional Notation for Peak Scaling

The process of peak scaling was described in section 4.3 and has not been altered apart from what was described in section 5 of this report. However, to better understand the concepts introduced in this chapter, some further definitions will be necessary.

Formally, we can, based on equation 4.7, describe the individual scaling of a peak matrix $\boldsymbol{P}_w = \begin{bmatrix}\boldsymbol{p}^{(1)} & \dots & \boldsymbol{p}^{(K)}\end{bmatrix}$, $\boldsymbol{P}_w \in \mathbb{R}^{3\times K}$ at time step w as a function $\Gamma\colon \mathbb{R}^{3\times K} \to \mathbb{R}^{3\times K}$ but with a number of side effects. For all of the operations presented in this section, it is however easier to describe this behaviour in terms of a scale operation of individual peaks, i.e. $\Gamma\colon \mathbb{R}^3 \to \mathbb{R}^3$ for each peak $\boldsymbol{p}^{(k)} \in \boldsymbol{P}_w$, $\boldsymbol{p}^{(k)} \in \mathbb{R}^3$ for $k = 1, \dots, K$. This function is at time step w defined as

$$\boldsymbol{p}^{(*)} = \Gamma\left(\boldsymbol{p}, t_w^{(\mathcal{Y})}, \mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}, \mathcal{H}_w^{(q)}\right), \qquad \forall\, \boldsymbol{p} \in \boldsymbol{P}_w \tag{6.1}$$

where $t_w^{(\mathcal{Y})} \in \mathbb{R}_+$ denotes the (scheduler's) time at generation time step w, $\mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}$ denotes the slice in the corpus corresponding to the time $t^{(\mathcal{Y})}$ of the peak (per equation 5.10) and $\mathcal{H}_w^{(q)}$ is the history of output slices from the previous $q \in \mathbb{Z}_+$ time steps, i.e.

$$\mathcal{H}_w^{(q)} = \left\{S_{w-1}^{(\mathcal{Y})}, \dots, S_{w-q}^{(\mathcal{Y})}\right\}, \tag{6.2}$$

where $\mathcal{S}_{w-m}^{(\mathcal{Y})}$ denotes the generated output m time steps ago.

This behaviour was previously handled at each level in the architecture by the `MergeAction`, which handled both merging peaks from multiple layers as well as scaling them individually, but has now been moved to the `ScaleAction` class and is only computed once at the top level (`Player` class)[1]. Do note that there are no limitations

[1] See [1] for context regarding these different classes

to the number of scale actions that can be applied, so the final value of each peak $\boldsymbol{p}^{(*)}$ can for J scale actions be defined as

$$\boldsymbol{p}^{(*)} = \left(\Gamma^{(J)} \circ \Gamma^{(J-1)} \circ \cdots \circ \Gamma^{(1)}\right)\left(\boldsymbol{p}, t_w^{(\mathcal{Y})}, \mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}, \mathcal{H}_w\right) \tag{6.3}$$

where $\circ$ denotes function composition. Each of the scale actions described in the following sections are optional and can be added/removed dynamically by the user.

6.2 Transposition Consistency

One concern related to the introduction of transpositions is that all transforms are given equal probability. In certain (tonal) cases, this may mean that unbiased and too frequent jumping between transpositions will cause issues between the original harmonic function of a particular chord and its harmonic function in the new real-time context. The Transposition Consistency scale action was introduced as an attempt to remedy this by simply scaling all peaks with a user controlled parameter $\gamma \in \mathbb{R}_{[0,1]}$ if the peak's transposition does not correspond to the transposition of the previous output slice, i.e.

$$\boldsymbol{p}^{(*)} = \begin{cases} \boldsymbol{p} & \text{if } \mathcal{T}_{w-1} = \mathcal{T}_{\boldsymbol{p}} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{p} & \text{otherwise} \end{cases} \tag{6.4}$$

where $\mathcal{T}_{w-1}$ denotes the transform of the previous output slice $\mathcal{S}_{w-1}^{(\mathcal{Y})}$ and $\boldsymbol{p}$ is defined (according to equation 5.10) as

$$\boldsymbol{p} = \begin{bmatrix} t_{\boldsymbol{p}}^{(\mathcal{C})} \\ y_{\boldsymbol{p}} \\ \mathcal{T}_{\boldsymbol{p}} \end{bmatrix}. \tag{6.5}$$

6.3 Taboo and Auto-Jump

Taboo and auto-jump are two behaviours that existed in earlier versions of Somax but were not originally included when the code base was rewritten in [3]. With this update, they have been reintroduced as scale actions in the new architecture. Taboo is a simple but efficient way to prevent a player from getting stuck in a short loop by reducing the score $y_{\boldsymbol{p}}$ of all peaks whose temporal coordinates $t_{\boldsymbol{p}}^{(\mathcal{C})}$ corresponds to the $q \in \mathbb{Z}_+$ (user-controlled parameter) most recent output slices to 0, i.e.

$$\boldsymbol{p}^{(*)} = \begin{cases} \begin{bmatrix} t_{\boldsymbol{p}}^{(\mathcal{C})} \\ 0 \\ \mathcal{T}_{\boldsymbol{p}} \end{bmatrix} & \text{if } \mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})} \in \mathcal{H}_w^{(q)} \\ \boldsymbol{p} & \text{otherwise.} \end{cases} \tag{6.6}$$

The purpose of the auto-jump scale action is to prevent too long sequences of the original corpus to be replicated in the output by gradually increasing the probability of a jump. More precisely, it will gradually decrease the score of peaks corresponding to the in the corpus following slice within a user-defined interval $[a, b], a, b \in \mathbb{Z}_+, b > a$

number of generated output events by a factor $\gamma \in \mathbb{R}_{[0,1]}$ calculated according to the following formula:

$$\gamma = \begin{cases} 1 & \text{if } \tau \leq a \\ 0.5^{\tau - a} & \text{if } a < \tau < b \\ 0 & \text{otherwise,} \end{cases} \tag{6.7}$$

where τ corresponds to the number of in the corpus consecutive slices computed by algorithm 2, where the function $u\left(\mathcal{S}^{(\mathcal{Y})}\right)$ returns the index $u \in [1, \ldots, U]$ of $\mathcal{S}^{(\mathcal{Y})}$ in its original corpus $\mathcal{C}$. The result of this calculation is only applied to the score of any

Algorithm 2 Computing number of consecutive slices τ

```
$\tau := 0$
$u := u\left(\mathcal{S}^{(\mathcal{Y})}_{w-1}\right)$
$i := 2$
**while** $u - u\left(\mathcal{S}^{(\mathcal{Y})}_{w-i}\right) = 1$ **do**
    $\tau := \tau + 1$
    $u := u\left(\mathcal{S}^{(\mathcal{Y})}_{w-i}\right)$
    $i := i + 1$
**end while**
```

peak corresponding to the in the corpus next consecutive slice, i.e.

$$\boldsymbol{p}^{(*)} = \begin{cases} \begin{bmatrix} t^{(\mathcal{C})}_{\boldsymbol{p}} \\ \gamma y_{\boldsymbol{p}} \\ \mathcal{T}_{\boldsymbol{p}} \end{bmatrix} & \text{if } u\left(\mathcal{S}^{(\mathcal{C})}_{\boldsymbol{p}}\right) = u\left(\mathcal{S}^{(\mathcal{Y})}_{w-1}\right) - 1 \\ \boldsymbol{p} & \text{otherwise.} \end{cases} \tag{6.8}$$

6.4 Parametric Filters

Somax has up to this point mainly been operating on two musical dimensions when selecting its output: chroma and pitch. Here, a number of scale actions are introduced to give the user some degree of control over other musical dimensions such as loudness, spectral distribution and various temporal aspects of the generated output. Similarly to other scale actions, all of these filters operate by scaling the score of the peak according to a parameter $\gamma \in \mathbb{R}_{[0,1]}$, i.e.

$$\boldsymbol{p}^{(*)} = \begin{bmatrix} t^{(\mathcal{C})} \\ \gamma y_{\boldsymbol{p}} \\ \mathcal{T}_{\boldsymbol{p}} \end{bmatrix} \tag{6.9}$$

where γ can be described as a function operating on a trait θ defined in the corpus. More specifically, each of those filters (apart from the last one) take the form of a gaussian defined so that

$$\gamma = \gamma\left(\theta, \bar{\theta}\right) = \exp\left[-\frac{(\theta - \bar{\theta})^2}{2s^2}\right] \tag{6.10}$$

where $\bar{\theta}$ (unless otherwise specified) defines the desired value for the targeted musical trait and s (in all cases) defines the desired spread of the targeted parameter, effectively controlling how steeply peaks corresponding to slices whose traits fall outside the requested value should be attenuated. Both $\bar{\theta}$ and s are (unless otherwise specified) user-controlled parameters. As Somax cannot construct corpora from audio files at the moment, all of these filters are only defined for corpora based on MIDI files.[2] Note that none of these scale actions will remove any peaks no matter how far away their corresponding traits fall from the requested value, they will at most drastically reduce their probability. The idea behind this is to to give the user control over the overall direction, where occasional fluctuations occur, ideally resulting in a more dynamic output. Also note that if no peaks exist for a given time step w, the scale action will not have any impact on the resulting output.

6.4.1 Loudness

The Loudness scale action allows the user to specify a requested maximum velocity within a slice. The parameters for γ are defined as

$$\theta = \max\left(\left\{\frac{\mathcal{N}\texttt{.velocity}}{128} \mid \mathcal{N} \in \mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}\right\}\right), \quad \theta \in R_{[0,1]} \tag{6.11}$$

i.e. the maximum normalized (MIDI-) velocity where $\mathcal{N}$ denotes each note within the corresponding slice $\mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}$, and $\bar{\theta} \in \mathbb{R}_{[0,1]}$ is the by the user requested normalized maximum velocity.

6.4.2 Vertical Density

The Vertical Density scale action allows the user to control how many notes there ideally should be in the output. The parameters for γ are given by

$$\theta = \left|\mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}\right|, \quad \theta \in \mathbb{Z}_+ \tag{6.12}$$

and $\bar{\theta} \in \mathbb{Z}_+$ is the by the user requested number of notes.

6.4.3 Duration

The Duration scale action gives the user control over the average duration of each slice in the output. γ is defined so that

$$\theta = d_{\boldsymbol{p}} \tag{6.13}$$

where $d_{\boldsymbol{p}} \in \mathbb{R}_+$ denotes the duration in ticks of slice $\mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}$ and $\bar{\theta} \in \mathbb{R}_+$ is user-defined.

6.4.4 Tempo Consistency

When using a corpus consisting of multiple sections with varying tempi, one problem is that the tempo changes too often, sometimes with every slice, which may obscure the user's perception of beat. The Tempo Consistency scale action is designed to remedy this behaviour. Unlike the previous filters in this section, it is not designed so that the

[2] but similar solutions exist in the audio domain and the architecture fully supports adding such solutions, should this change in the future.

user defines a requested tempo (as this can already be done in the scheduler), rather it serves as a low-pass filter based on the history of previous tempi so that

$$\theta = \zeta_{\boldsymbol{p}}, \quad \theta \in \mathbb{R}_{+} \tag{6.14}$$

$$\bar{\theta} = \frac{1}{q}\sum_{k=1}^{q} \zeta_{w-q}, \quad \bar{\theta} \in \mathbb{R}_{+} \tag{6.15}$$

where $\zeta_{\boldsymbol{p}}$ denotes the tempo in BPM of slice $\mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}$, ζ_{w-q} denotes the tempo of previous output slice $\mathcal{S}_{w-q}^{(\mathcal{Y})}$ and q is a user controlled parameter corresponding to the length of the filter, i.e. how many previous output slices to take into account when applying the filter.

6.4.5 Spectral Distribution

The final filter in this category, the Spectral Distribution scale action, is intended to give the user some control over the spectral distribution, which in this case corresponds to the number of MIDI notes in each octave, and is the only filter in this category that does not apply the above procedure of scaling based on a gaussian. Here $\gamma \in \mathbb{R}_{[0,1]}$ is defined as a function $\gamma\left(\boldsymbol{\theta}, \bar{\boldsymbol{\theta}}\right)$ where the user specifies a vector $\hat{\boldsymbol{\theta}} \in \mathbb{R}_{[0,1]}^{11}$ where the value at $\hat{\boldsymbol{\theta}}[i]$ corresponds to the weight to apply to the i:th octave, $i \in [0, 11)$.[3] A similar value $\boldsymbol{\theta} \in \mathbb{R}_{[0,1]}^{11}$ is calculated for each slice $\mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}$ corresponding to a peak using algorithm 3:

Algorithm 3 Computing spectral distribution $\boldsymbol{\theta}$ for slice $\mathcal{S}_{\boldsymbol{p}}^{(\mathcal{S})}$

```
$\boldsymbol{\theta} := \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}$
**for** $\mathcal{N} \in \mathcal{S}_{\boldsymbol{p}}^{(\mathcal{C})}$ **do**
  $b := \left\lfloor \frac{\mathcal{N}.\texttt{pitch}}{12} \right\rfloor$
  $\boldsymbol{\theta}[b] := \boldsymbol{\theta}[b] + 1$
**end for**
$\boldsymbol{\theta} := \left(\max_{\theta \in \boldsymbol{\theta}} \theta\right)^{-1} \boldsymbol{\theta}$
```

Finally, γ is given by the Euclidean distance between the requested distribution and the actual, i.e.

$$\gamma\left(\boldsymbol{\theta}, \bar{\boldsymbol{\theta}}\right) = \left\lVert \boldsymbol{\theta} - \bar{\boldsymbol{\theta}} \right\rVert_2 . \tag{6.16}$$

6.5 Regional Masking

The final scale action is the Regional Masking scale action, which allows the user to specify a region of indices within the corpus that the output should be selected from,

[3] Octave numbers start from MIDI note number 0, e.g. octave 0 corresponds to note numbers 0-11, octave 1 note numbers 12-23, etc., resulting in a total of 11 octaves, even if the bottom and top octaves are very rarely used in musical contexts.

e.g. to select only one of multiple movements in a multi-movement corpus. More specifically, the user specifies an interval $[a, b],\ a, b \in \mathbb{Z}_{[1,U]},\ a < b$ where U denotes the size of corpus $\mathcal{C}$, and applies a parameter γ (according to equation 6.9) defined as

$$\gamma = \begin{cases} 1 & \text{if } a \leq u\left(\mathcal{S}_{\boldsymbol{p}}^{(C)}\right) \leq b \\ 0 & \text{otherwise.} \end{cases} \tag{6.17}$$

6.6 Output Selection Modes

The final step in the entire generation process is as described in section 4 to select an output slice $\mathcal{S}_w^{(\mathcal{Y})}$ from the scaled peak matrix $\boldsymbol{P}_w^{(*)}$ at time step w. This was in section 4.4 defined as the slice corresponding to the peak with the highest score (or if no peaks exist, simply the slice in the corpus following the previous output slice), but can be generalized to a selection function $\Omega \colon \mathbb{R}^{3\times K} \to \mathcal{S}$ defined as

$$\mathcal{S}_w^{(\mathcal{Y})} = \Omega\left(\boldsymbol{P}_w^{(*)}, \mathcal{C}, \mathcal{H}_w\right). \tag{6.18}$$

This report introduces two new modes for output selection: Probabilistic output selection and Quality Threshold. To provide a better framework for the new classifiers, we will briefly revisit this default output selector and introduce some additional definitions, followed by the introduction of the new modes.

6.6.1 Output Selection in the Generalized Case

As per equation 6.18, we define the output slice $\mathcal{S}_w^{(\mathcal{Y})}$ at time step w as a function of the peak matrix $\boldsymbol{P}_w^{(*)}$, the corpus $\mathcal{C}$ and the history of previous output slices $\mathcal{H}_w$. More specifically, we can for all output selection modes define a set $\Xi = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_H\}$ of $H \in \mathbb{Z}_*$ viable peak candidates with the corresponding set of slices $\xi = \left\{\mathcal{S}_{\boldsymbol{p}_1}^{(\mathcal{C})}, \ldots, \mathcal{S}_{\boldsymbol{p}_H}^{(\mathcal{C})}\right\}$. For each slice, a probability ρ_h is defined so that

$$\rho_h = P\left(\mathcal{S}_{\boldsymbol{p}_h}^{(\mathcal{C})}\right), \quad \forall\, \mathcal{S}_{\boldsymbol{p}_h}^{(\mathcal{C})} \in \xi. \tag{6.19}$$

Given a stochastic variable $X \sim \mathrm{U}[0, 1]$ the output slice is selected through a sampling function $g(\rho)$ so that

$$g(\rho) = \begin{cases} \mathcal{S}_{\boldsymbol{p}_1}^{(\mathcal{C})} & \text{if } X < \rho_1 \\ \mathcal{S}_{\boldsymbol{p}_2}^{(\mathcal{C})} & \text{if } \rho_1 \leq X < \rho_2 \\ \vdots & \\ \mathcal{S}_{\boldsymbol{p}_{H-1}}^{(\mathcal{C})} & \text{if } \rho_{H-2} \leq X < \rho_{h-1} \\ \mathcal{S}_{\boldsymbol{p}_H}^{(\mathcal{C})} & \text{otherwise} \end{cases} \tag{6.20}$$

In the case of the default output selection mode («Maximum»), we define Ξ as the set of peaks corresponding to the peak with the highest score ($H > 1$ if multiple peaks have the same maximum value), i.e.

$$\Xi = \left\{\boldsymbol{p} \,\middle|\, \boldsymbol{p} \in \boldsymbol{P}_w^{(*)} \wedge \left(y_{\boldsymbol{p}} = \max_{\boldsymbol{p}_j \in \boldsymbol{P}_w^{(*)}} y_{\boldsymbol{p}_j}\right)\right\}, \tag{6.21}$$

with corresponding probabilities

$$\rho_h = P\left(\mathcal{S}^{(\mathcal{C})}_{\boldsymbol{p}_h}\right) = \frac{1}{|\Xi|}, \quad \forall\, \mathcal{S}^{(\mathcal{C})}_{\boldsymbol{p}_h} \in \xi. \tag{6.22}$$

The output slice $\mathcal{S}^{(\mathcal{Y})}_w$ is given by

$$\Omega\left(\boldsymbol{P}^{(*)}_w, \mathcal{C}, \mathcal{H}_w\right) = \begin{cases} g(\rho) & \text{if } \Xi \neq \varnothing \\ \mathcal{S}^{(\mathcal{C})}_{u+1} & \text{otherwise,} \end{cases} \tag{6.23}$$

where $g(\rho)$ is defined according to equation 6.20 and $\mathcal{S}^{(\mathcal{C})}_{u+1}$ denotes the slice following the previous output slice, i.e. the slice $\mathcal{S}^{(\mathcal{C})} \in \mathcal{C}$ fulfilling the condition

$$u\left(\mathcal{S}^{(\mathcal{C})}\right) = u\left(\mathcal{S}^{(\mathcal{Y})}_{w-1}\right) + 1 \qquad \operatorname{mod} U. \tag{6.24}$$

6.6.2 Probabilistic Output Selection

The Probabilistic output selection mode will, unlike the default output selection mode, take all peaks into account and create a probability density function where the probability is defined by the score of each peak, i.e.

$$\Xi = \left\{\boldsymbol{p} \,\middle|\, \boldsymbol{p} \in \boldsymbol{P}^{(*)}_w\right\} \tag{6.25}$$

and

$$\rho_h = P\left(\mathcal{S}^{(\mathcal{C})}_{\boldsymbol{p}_h}\right) = \frac{y_{\boldsymbol{p}_h}}{\sum_{i=1}^{H} y_{\boldsymbol{p}_i}}, \tag{6.26}$$

where $g(\rho)$ and Ω are defined according to equations 6.20 and 6.23 respectively. In this mode, the probability of matches with lower quality (i.e. peaks with a lower score) increases, but the variation increases. Due to the latter, it's less likely to get stuck in deterministic loops when interacting with the system.

6.7 Quality Threshold

The Quality Theshold output selection mode is an extension that can be combined with either of the two previous modes, in which the user defines a threshold $\psi \in \mathbb{R}_+$, where all peaks with scores below this value will be discarded, i.e.

$$\Xi = \left\{\boldsymbol{p} \,\middle|\, \boldsymbol{p} \in \boldsymbol{P}^{(*)}_w \wedge \left(y_{\boldsymbol{p}} = \max_{\boldsymbol{p}_j \in \boldsymbol{P}^{(*)}_w} y_{\boldsymbol{p}_j}\right)\right\} \cap \left\{\boldsymbol{p} \,\middle|\, \boldsymbol{p} \in \boldsymbol{P}^{(*)}_w \wedge y_{\boldsymbol{p}} \geq \psi\right\} \tag{6.27}$$

when combined with the default (≪Maximum≫) mode and

$$\Xi = \left\{\boldsymbol{p} \,\middle|\, \boldsymbol{p} \in \boldsymbol{P}^{(*)}_w\right\} \cap \left\{\boldsymbol{p} \,\middle|\, \boldsymbol{p} \in \boldsymbol{P}^{(*)}_w \wedge y_{\boldsymbol{p}} \geq \psi\right\} \tag{6.28}$$

when combined with the probabilistic mode. The output selection function is in both cases defined as

$$\Omega\left(\boldsymbol{P}^{(*)}_w, \mathcal{C}, \mathcal{H}_w\right) = \begin{cases} g(\rho) & \text{if } \Xi \neq \varnothing \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{6.29}$$

where ρ is defined as equations 6.22 and 6.26 respectively and $g(\rho)$ as equation 6.20. More importantly, as seen in equation 6.29, the output is undefined if no peaks exists in Ξ, either because no matches were found in earlier stages of the generation process so that $\boldsymbol{P}_w^{(*)}$ is empty or because no peaks fulfil the condition $\boldsymbol{y_p} \geq \psi$. When undefined, no output will be generated for this particular generation step w. In other words, if the quality of the slices corresponding to the existing peaks does not match the requested quality (as defined by the threshold ψ), no output will be generated at all, hence introducing a degree of sparsity in the generated material.

Bibliography

[1] The Somax Software Architecture. Ircam-STMS, Technical Report, 2021.

[2] Laurent Bonnasse-Gahot. An update on the SOMax project. *Ircam-STMS, Tech. Rep*, 2014.

[3] Joakim Borg. Somax 2: A Real-time Framework for Human-Machine Improvisation. *Internal Report - Aalborg University Copenhagen*, 2019.

[4] Daniel PW Ellis. Beat tracking by dynamic programming. *Journal of New Music Research*, 36(1):51–60, 2007.

[5] Peter Grosche and Meinard Muller. Extracting predominant local pulse information from music recordings. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(6):1688–1701, 2010.