stochastica organica

score/instructions

k. hagan 2006

Introduction

There are two parts to *stochastica organica*: the patch and the realization of the work as I have composed it. Although the patch has been programmed to generate the original work, it may also be used to create other works. This score details both the procedure for performing *stochastica organica* and the details of the patch so that it may be used for other purposes.

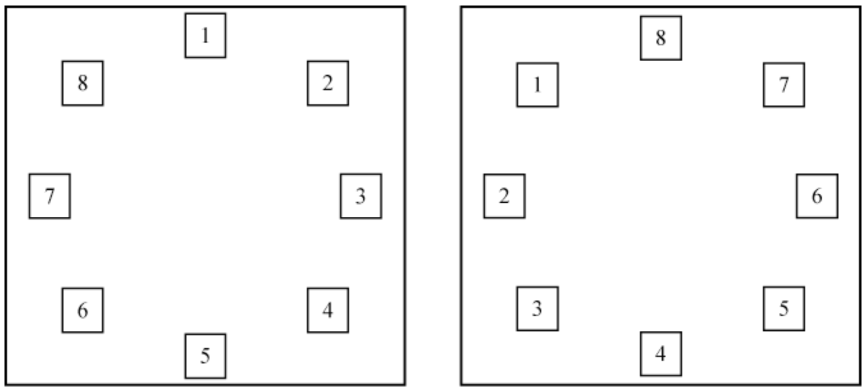
Pd can be run across many platforms. Depending on the platform, different options may be found in different menus. This patch was developed using Pd 0.38-3 for Macintosh. I use Macintosh nomenclature (i.e., folder) for directories and menus.

Set-up

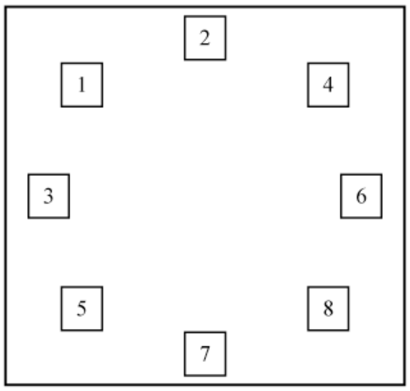
The folder containing the patches and subpatches must not be changed, or the objects in the patch may not be recognized. The sound sources must remain in the subfolder, **sounds**.

1. Start Pd. In the **Pd** menu, choose **Preferences**, and **Audio Settings…** .
2. Set the output device to your multi-channel interface.
3. Enter 8 channels in the output. Click **Apply** and close the window.
4. Go to the **Media** menu and choose **Test Audio and MIDI…**. Make sure that all channels are working. Assign the channels in a circle. The placement of channel 1 and the direction of the circle are not important. (See the figure below).

   Note: For non-circular arrangements, the speaker locations can be changed in the **init** subpatch. But, it is more efficient to reassign channels at the hardware level.



Two acceptable arrangements



Spatialization will not work
in this arrangement.

Figure 1: Proper speaker assignments

Playing *stochastica organica*

1. Open the patch by opening **___start_multichannel.pd**.
2. Make sure the compute audio box is checked in the main Pd window.
3. Find the **output~** object on the main page. Set the output dB to approximately 95 in the box labeled **dB**. This may be adjusted as needed for the room and equipment.
4. In the **controls** object, click the button labeled **start**. The piece will run automatically.
5. When the piece is over, click the button labeled **stop** to stop the counter.

Recording

1. Follow steps 1 – 3 above.
2. Find the **writesf~** object near the **output~** object. Click the message: **open ./performance.aiff**
3. Click the **start** message immediately below the **open** message.
4. In the **controls** object, click the button labeled **start**.
5. When the piece is over, click the button labeled **stop**.
6. Find the **stop** message connected to the **writesf~** object and click it to automatically save the recording to a file called **performance.aiff** found in the patch's folder.

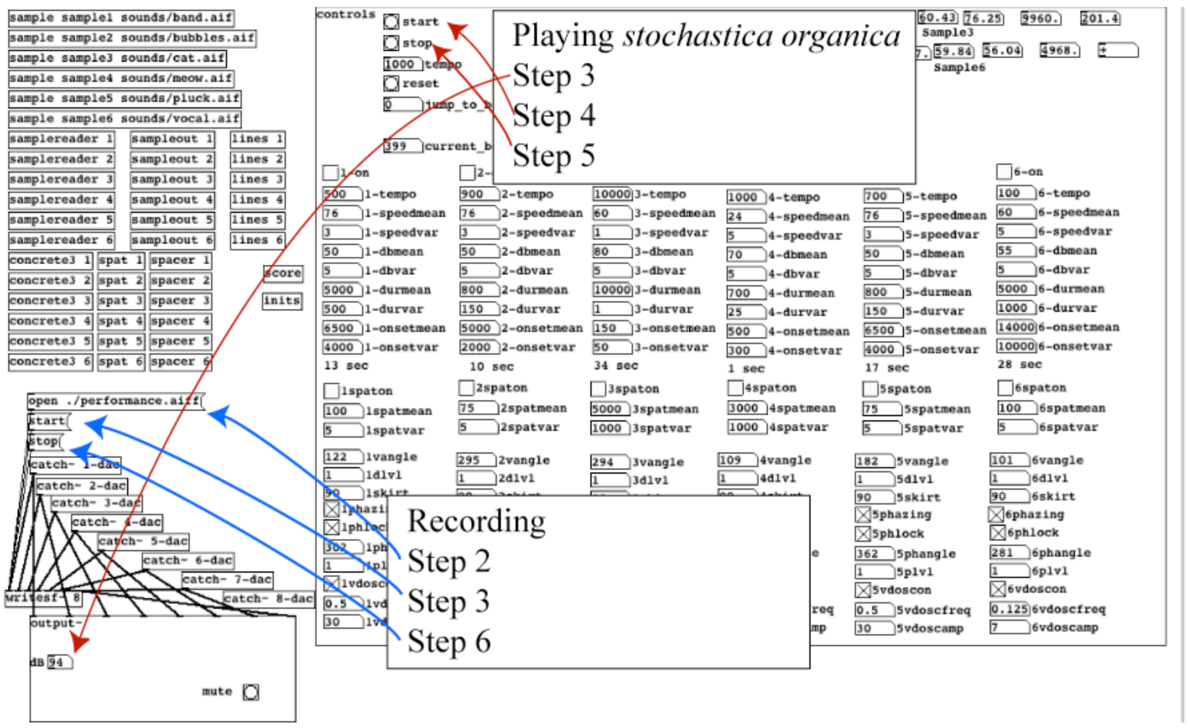


Figure 2: Locations of objects for playback and recording steps

Patch structure

The patch can be split into three main sections: the objects that send the parameter values (control objects), the objects that generate the random variables to control the samples and the spatialization (random generators), and the playback mechanisms that perform the samples in a virtual location (sample playback). Outputs are both the 8-channel output and a recording object.

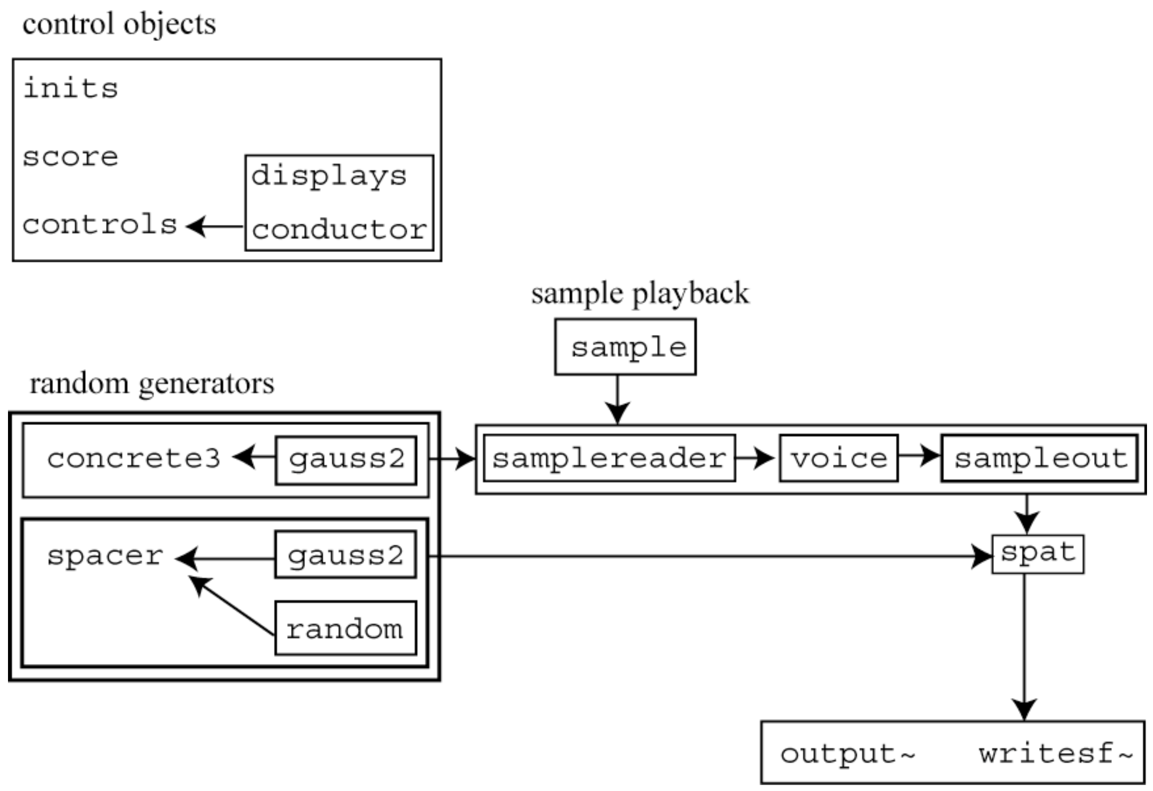


Figure 3: Patch structure

Main controls

The **controls** subpatch displays and sends parameter values. The **lines** objects will take parameter values and ramp them. Each line is named by the parameter with "l" (for line) at the end. For example, to ramp **1spatmean** to 1000 in 500 milliseconds, send the message "1000 500" to **1spatmeanl**. The control labeled **1spatmean** will display the values as they ramp.

Each control is a parameter sent to the random number generators or spatial controls. The following sections explain the parameters.

concrete3

concrete3 is named as a nod to *musique concrète*. In original versions, **concrete** utilized uniformly distributed random variables to sample from a soundfile. In **concrete3**, the generators are all Gaussian distributions with a specified mean and variance. Given a soundfile, random variable generators choose the onset point (**onsetmean** and **onsetvar**), the duration of the sample (**durmean** and **durvar**), the loudness (**dbmean** and **dbvar**), and the playback speed (**speedmean** and **speedvar**). The playback speed is represented in MIDI. No transposition is equivalent to 60 (middle C). Twice as fast is one octave above (72); twice as slow is one octave below (48).

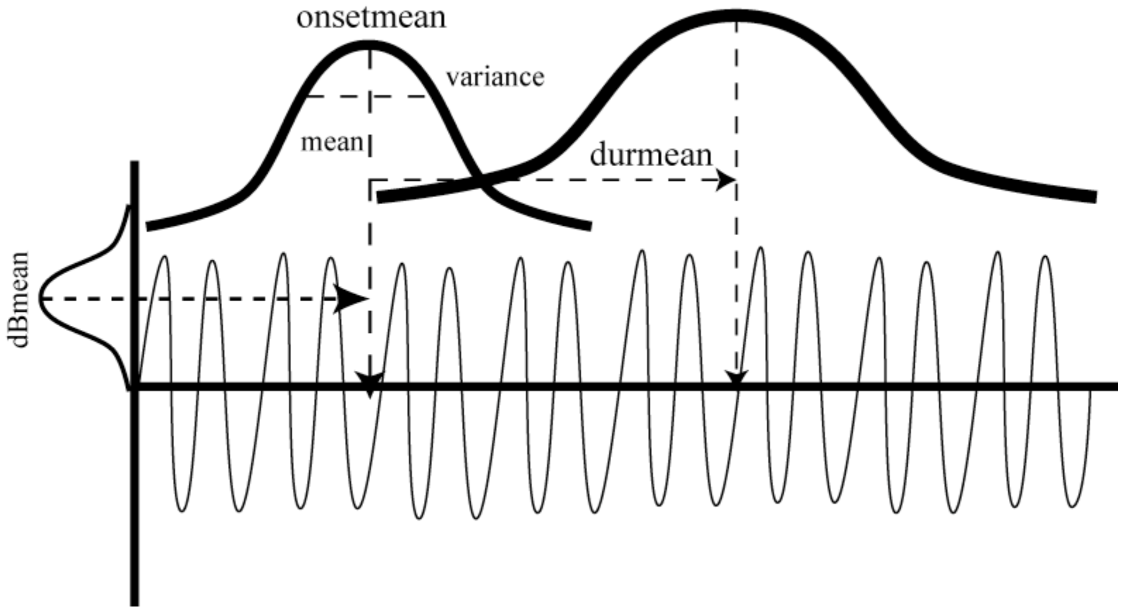
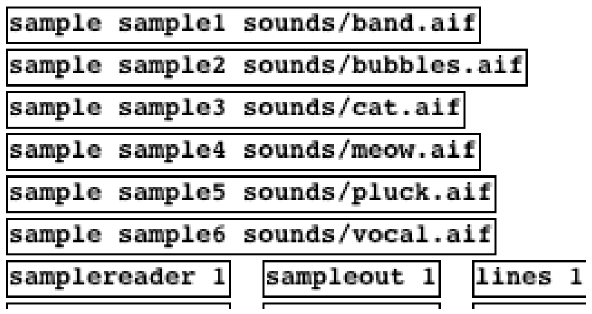


Figure 4: Sampling variables in relation to a soundfile

Any soundfile can be used by typing the soundfile path in the **sample** object. This pre-loads a table containing the soundfile that is sampled by **concrete3**. One word of caution: make sure that the **onsetmean**, **onsetvar**, **durmean**, and **durvar** variables cannot go beyond the actual soundfile length or very loud, very unpleasant garbage will result.



```
sample sample1 sounds/band.aif
sample sample2 sounds/bubbles.aif
sample sample3 sounds/cat.aif
sample sample4 sounds/meow.aif
sample sample5 sounds/pluck.aif
sample sample6 sounds/vocal.aif
samplereader 1   sampleout 1   lines 1
```

Figure 5: Soundfiles are stored in the sample object

spacer

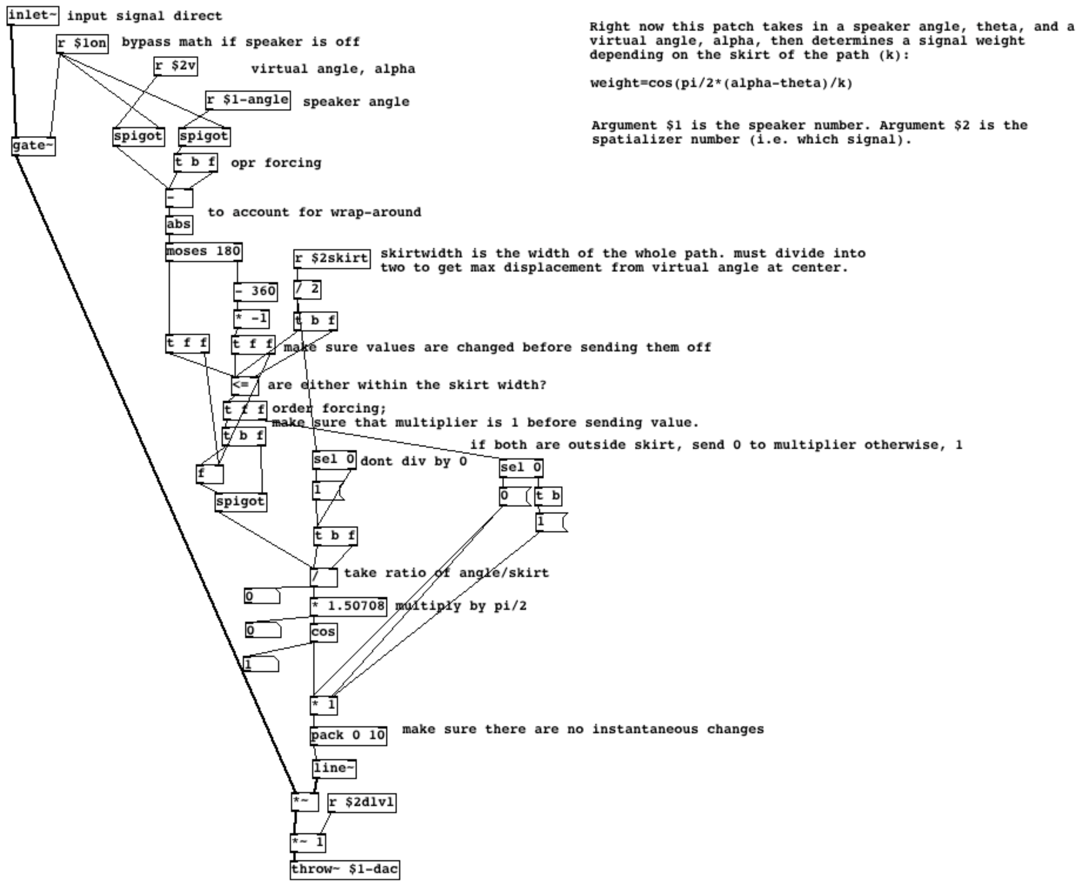
The spacer subpatch chooses a location in 360 degrees by a uniformly distributed variable. It is coupled with a time in which the sound moves to that location from its current location. The time is determined by a Gaussian distribution (**spatmean** and **spatvar**). The spacer subpatch sends the location as the virtual angle of the sound (**vangle**) to the **spat** object.

spat

The **spat** subpatch works on a psychoacoustic principle. Two identical signals in opposing speakers will appear to come from the closer speaker. If one copy of the signal moves in and out of phase, the sound will seem to jump between the two speakers.

A direct signal is sent to a virtual location specified by **vangle**. A copy can be locked (**phlock**) at 180 degrees to **vangle** (**phangle**). If a copied signal is then oscillated (**vdoscon**) through a delay ranging from 0 to around 5 milliseconds (**vdoscamp**), the copy moves in and out of phase with the original.

The weight of the signal and its phased copy in each speaker is calculated by the **weightor~** and **phweightor~** objects. These objects need a “skirt” (the width of a signal around the virtual angle) and the physical location of each speakers (set in the **init** subpatch). Each **weightor~** then calculates the power of the signal based on the speaker’s location in the circle, the virtual angle of the signal, and the skirt of the signal.


inlet~
input signal direct
r $1on
bypass math if speaker is off
r $2v
virtual angle, alpha
r $1-angle
speaker angle
gate~
spigot
spigot
t b f
opr forcing
-
to account for wrap-around
abs
moses 180
r $2skirt
- 360
/ 2
* -1
t b f
t f f
t f f
make sure values are changed before sending them off
<=
are either within the skirt width?
t f f
order forcing;
make sure that multiplier is 1 before sending value.
t b f
if both are outside skirt, send 0 to multiplier otherwise, 1
sel 0
dont div by 0
sel 0
f
1
0
t b
spigot
1
t b f
/
take ratio of angle/skirt
0
* 1.50708
multiply by pi/2
0
cos
1
* 1
pack 0 10
make sure there are no instantaneous changes
line~
*~
r $2dlvl
*~ 1
throw~ $1-dac

Right now this patch takes in a speaker angle, theta, and a virtual angle, alpha, then determines a signal weight depending on the skirt of the path (k):

weight=cos(pi/2*(alpha-theta)/k)

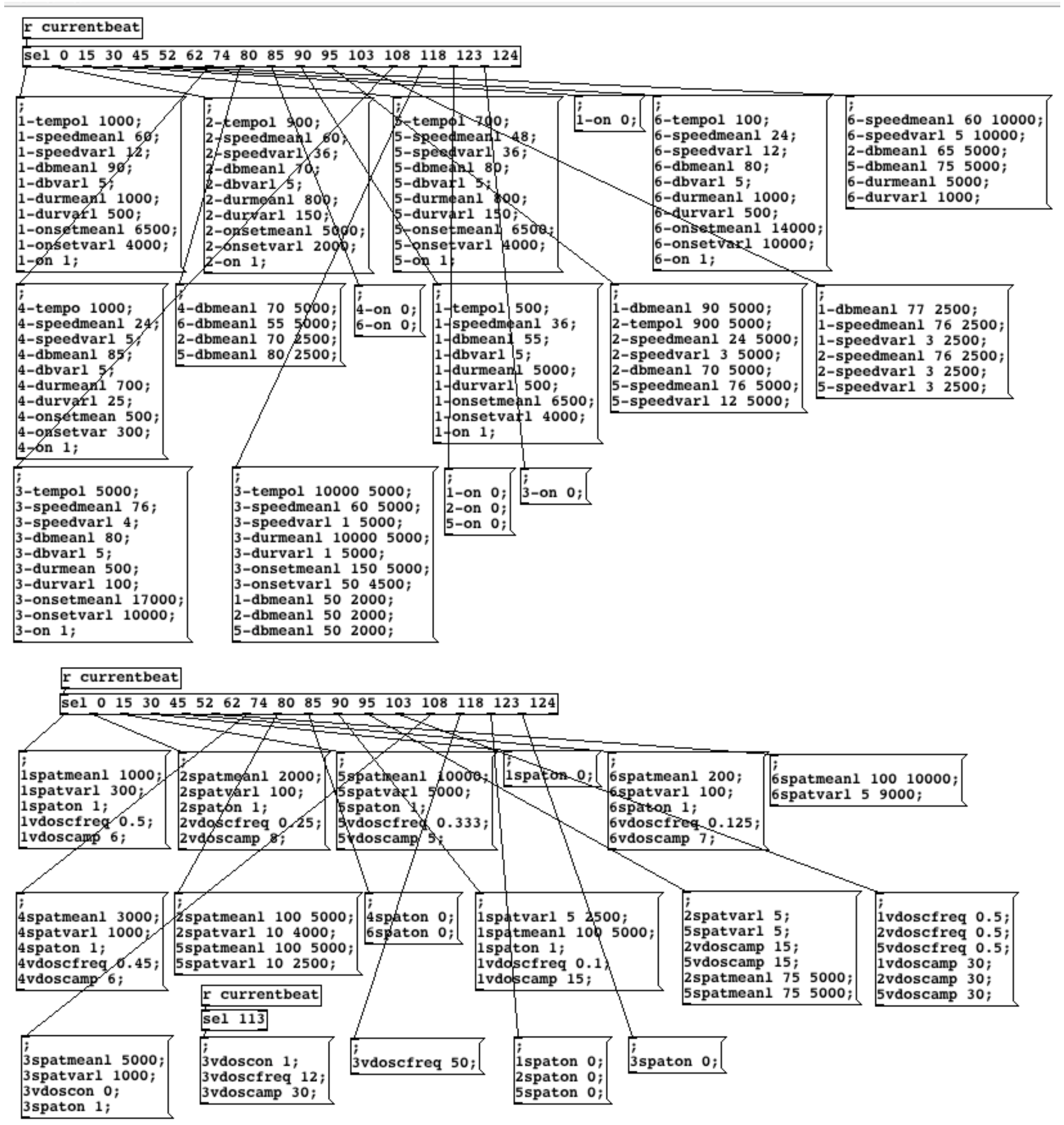
Argument $1 is the speaker number. Argument $2 is the spatializer number (i.e. which signal).

skirtwidth is the width of the whole path. must divide into two to get max displacement from virtual angle at center.

Figure 6: The weightor~ object and its calculations for speaker power

score

The best way to see how each parameter effects the playback is to look at the score of *stochastica organica*. The conductor object (in **controls**) is a simple counter increasing per second. The score sends the appropriate values at the specified seconds.



```
r currentbeat
sel 0 15 30 45 52 62 74 80 85 90 95 103 108 118 123 124

;
1-tempol 1000;
1-speedmeanl 60;
1-speedvarl 12;
1-dbmeanl 90;
1-dbvarl 5;
1-durmeanl 1000;
1-durvarl 500;
1-onsetmeanl 6500;
1-onsetvarl 4000;
1-on 1;

;
2-tempol 900;
2-speedmeanl 60;
2-speedvarl 36;
2-dbmeanl 70;
2-dbvarl 5;
2-durmeanl 800;
2-durvarl 150;
2-onsetmeanl 5000;
2-onsetvarl 2000;
2-on 1;

;
5-tempol 700;
5-speedmeanl 48;
5-speedvarl 36;
5-dbmeanl 80;
5-dbvarl 5;
5-durmeanl 800;
5-durvarl 150;
5-onsetmeanl 6500;
5-onsetvarl 4000;
5-on 1;

;
1-on 0;

;
6-tempol 100;
6-speedmeanl 24;
6-speedvarl 12;
6-dbmeanl 80;
6-dbvarl 5;
6-durmeanl 1000;
6-durvarl 500;
6-onsetmeanl 14000;
6-onsetvarl 10000;
6-on 1;

;
6-speedmeanl 60 10000;
6-speedvarl 5 10000;
2-dbmeanl 65 5000;
5-dbmeanl 75 5000;
6-durmeanl 5000;
6-durvarl 1000;

;
4-tempo 1000;
4-speedmeanl 24;
4-speedvarl 5;
4-dbmeanl 85;
4-dbvarl 5;
4-durmeanl 700;
4-durvarl 25;
4-onsetmean 500;
4-onsetvar 300;
4-on 1;

;
4-dbmeanl 70 5000;
6-dbmeanl 55 5000;
2-dbmeanl 70 2500;
5-dbmeanl 80 2500;

;
4-on 0;
6-on 0;

;
1-tempol 500;
1-speedmeanl 36;
1-dbmeanl 55;
1-dbvarl 5;
1-durmeanl 5000;
1-durvarl 500;
1-onsetmeanl 6500;
1-onsetvarl 4000;
1-on 1;

;
1-dbmeanl 90 5000;
2-tempol 900 5000;
2-speedmeanl 24 5000;
2-speedvarl 3 5000;
2-dbmeanl 70 5000;
5-speedmeanl 76 5000;
5-speedvarl 12 5000;

;
1-dbmeanl 77 2500;
1-speedmeanl 76 2500;
1-speedvarl 3 2500;
2-speedmeanl 76 2500;
2-speedvarl 3 2500;
5-speedvarl 3 2500;

;
3-tempol 5000;
3-speedmeanl 76;
3-speedvarl 4;
3-dbmeanl 80;
3-dbvarl 5;
3-durmean 500;
3-durvarl 100;
3-onsetmeanl 17000;
3-onsetvarl 10000;
3-on 1;

;
3-tempol 10000 5000;
3-speedmeanl 60 5000;
3-speedvarl 1 5000;
3-durmeanl 10000 5000;
3-durvarl 1 5000;
3-onsetmeanl 150 5000;
3-onsetvarl 50 4500;
1-dbmeanl 50 2000;
2-dbmeanl 50 2000;
5-dbmeanl 50 2000;

;
1-on 0;
2-on 0;
5-on 0;

;
3-on 0;

r currentbeat
sel 0 15 30 45 52 62 74 80 85 90 95 103 108 118 123 124

;
1spatmeanl 1000;
1spatvarl 300;
1spaton 1;
1vdoscfreq 0.5;
1vdoscamp 6;

;
2spatmeanl 2000;
2spatvarl 100;
2spaton 1;
2vdoscfreq 0.25;
2vdoscamp 8;

;
5spatmeanl 10000;
5spatvarl 5000;
5spaton 1;
5vdoscfreq 0.333;
5vdoscamp 5;

;
1spaton 0;

;
6spatmeanl 200;
6spatvarl 100;
6spaton 1;
6vdoscfreq 0.125;
6vdoscamp 7;

;
6spatmeanl 100 10000;
6spatvarl 5 9000;

;
4spatmeanl 3000;
4spatvarl 1000;
4spaton 1;
4vdoscfreq 0.45;
4vdoscamp 6;

;
2spatmeanl 100 5000;
2spatvarl 10 4000;
5spatmeanl 100 5000;
5spatvarl 10 2500;

;
4spaton 0;
6spaton 0;

;
1spatvarl 5 2500;
1spatmeanl 100 5000;
1spaton 1;
1vdoscfreq 0.1;
1vdoscamp 15;

;
2spatvarl 5;
5spatvarl 5;
2vdoscamp 15;
5vdoscamp 15;
2spatmeanl 75 5000;
5spatmeanl 75 5000;

;
1vdoscfreq 0.5;
2vdoscfreq 0.5;
5vdoscfreq 0.5;
1vdoscamp 30;
2vdoscamp 30;
5vdoscamp 30;

r currentbeat
sel 113

;
3spatmeanl 5000;
3spatvarl 1000;
3vdoscon 0;
3spaton 1;

;
3vdoscon 1;
3vdoscfreq 12;
3vdoscamp 30;

;
3vdoscfreq 50;

;
1spaton 0;
2spaton 0;
5spaton 0;

;
3spaton 0;
```

Figure 7: The automatic score for *stochastica organica*