Programming a Music Synthesizer through Data Mining

Jörn Loviscach
Hochschule Bremen (University of Applied Sciences)
Flughafenallee 10
28199 Bremen, Germany
joern.loviscach@hs-bremen.de

ABSTRACT

Sound libraries for music synthesizers easily comprise one thousand or more programs ("patches"). Thus, there are enough raw data to apply data mining to reveal typical settings and to extract dependencies. Intelligent user interfaces for music synthesizers can be based on such statistics. This paper proposes two approaches: First, the user sets any number of parameters and then lets the system find the nearest sounds in the database, a kind of patch autocompletion. Second, all parameters are "live" as usual, but turning one knob or setting a switch will also change the settings of other, statistically related controls. Both approaches can be used with the standard interface of the synthesizer. On top of that, this paper introduces alternative or additional interfaces based on data visualization.

Keywords

Information visualization, mutual information, intelligent user interfaces

1. INTRODUCTION

Most software-based synthesizers adhere to standard programming interfaces to retrieve program data, to set parameters, and to notify other software if a parameter is changed through the synthesizer's graphical user interface. Thus, by creating appropriate host software, one can not only collect the sound programs for data mining, but also intervene in the actions triggered by the synthesizer's switches, knobs and sliders. The novel interfaces proposed in the paper employ this possibility to enhance a synthesizer's standard user interface through statistical data:

Patch Autocompletion. One may set only a selection of parameters such as the oscillators' pitch and waveforms and the envelope, but leave all other parameters untouched. Then the system can search for sound programs in the database in which the touched parameters have similar settings. This process is similar to word autocompletion in text-processing software. In this mode, the standard user interface of the synthesizer is augmented by an interactive parallel coordinates visualization.

Co-Variation. Whenever the user edits a parameter, other parameters are varied along with the first parameter according to their statistical relation with it. For instance, increasing the attack time for the amplitude envelope may also increase the attack time of the filter envelope. Setting an oscillator to a pulse wave may configure the LFO for pulse width modulation. In this mode, the standard user interface of the synthesizer is augmented by an arrangement of the parameters as dots on a 2D field. Statistically related parameters are placed next to each other. Every parameter influences its neighbors according to their distance and the joint statistics.

The presented methods work with synthesizers of the classic Moog type. Due to the large variations in their wave forms, sampling synthesizers may not benefit from the presented methods. Also synthesizers of the FM type cannot be treated well, since the acoustical meaning of their parameters changes drastically with the choice of the FM algorithm, that is: the interconnection of the operators.

The freely available software synthesizer Synth1 (`http://www.geocities.jp/daichi1969/softsynth/`) serves as a model for the experiments. It is implemented as a VST plug-in (`http://www.steinberg.de/324_1.html`) and offers 87 patch parameters; its sound library as collected from different sources on the Internet comprises 1250 patches including lead synth, pad, and effect sounds. The prototype software employs Hermann Seib's publically available C++ source code for a VST host program (`http://www.hermannseib.com/english/vsthost.htm`). The host code has been extended to communicate via Internet Protocol (IP) and to automatically extract all available patch data of the synthesizer and write them to a text file on startup. For easier visualization and debugging, the statistical computations and the augmented user interfaces are created in a C#-based application that reads in the text file with the patch data and sends and receives parameter change commands to and from the VST host through a local IP connection on the same computer, see Figure 1.

2. RELATED WORK

Statistical methods have a long history in sound and music computing, in particular concerning information extraction [5]. Attempts to learn about the human perception of timbre [3] are psychoacoustic studies and thus are inherently of a statistical nature. The approach proposed in this paper studies bypasses psychoacoustics, however, and directly evaluates the statistics of a sound library. One may liken this to the technical analysis of share prices, through which analysts try to learn about a company without taking a look at its fundamental data.

Several works that aim at handling the vast parameter spaces of synthesizers employ evolutionary methods with a human in the loop. Both Genophone [9] and MutaSynth [4] do not rely on detailed knowledge of the inner workings of the synthesis unit and thus can be used with a large

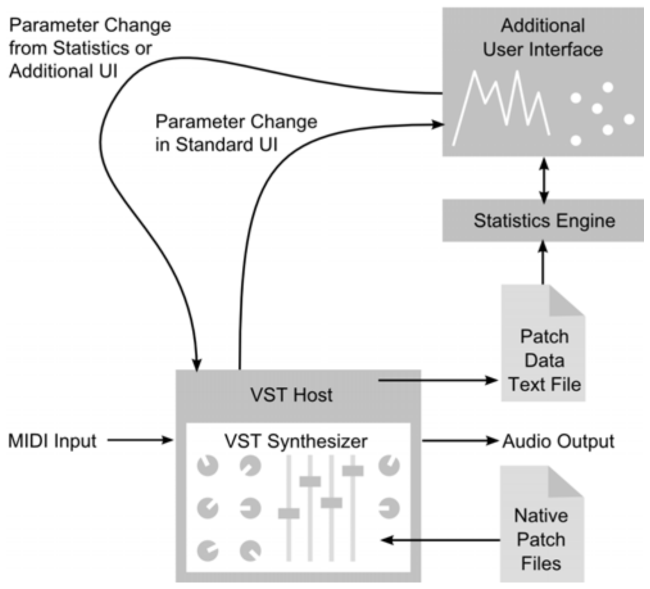
Parameter Change from Statistics or Additional UI
Parameter Change in Standard UI
Additional User Interface
Statistics Engine
Patch Data Text File
VST Host
VST Synthesizer
MIDI Input
Audio Output
Native Patch Files

Figure 1: The prototype comprises a modified VST host, additional graphical user interfaces, and a statistics engine.

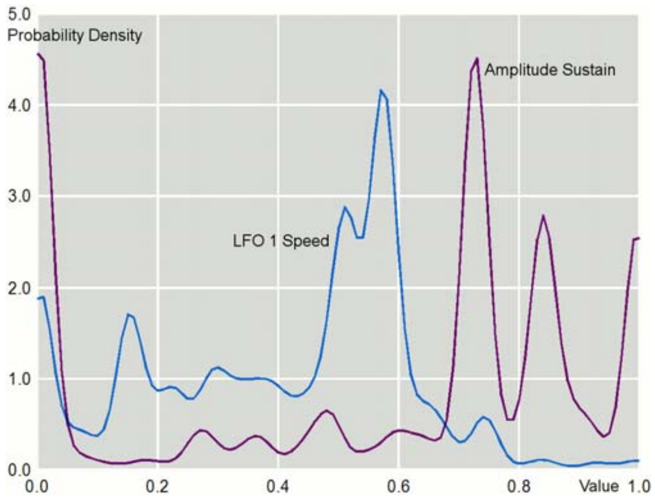

Figure 2: Most parameters possess a clustered distribution. Every parameter is normalized to the interval $[0, 1]$ and smoothed with a Parzen window of width 0.02.

range of sound generators—an aspect in which they resemble the approach presented in this paper. Hoffman and Cook [7] discuss a database model for feature-based synthesis. They employ an L^n distance in the low-dimensional feature space. The database retrieval is related to the autocompletion mode presented in this work, even though the databases' contents are different in the two works.

Co-variation, the second interaction mode described in this paper, can be interpreted as equipping a synthesizer with "embedded" metaparameters: Every single of the usual controls acts like a metaparameter in that it controls a number of related parameters. There is no dedicated control for the metaparameter's value, as opposed to standard approaches to metaparameters such as in VirSyn miniTERA `http://www.virsyn.de/`, in which all technical details are subsumed under a handful of general controls. Johnson and Gounaropoulos [8] train a neural net to map metaparameters to low-level controls.

Bencina [1] maps an arbitrary number of control dimensions to a 2D surface. This seems to be related to the covariation interaction mode described in this paper. Bencina, however, places parameter *settings* manually in 2D, whereas this work automatically places *parameters* as such.

3. PATCH AUTOCOMPLETION

Autocompletion is a standard feature in text-based software. While the user types a word, the software queries a dictionary of common terms and—if successful—either offers a context menu containing all possible completions or offers the shortest found completion for immediate insertion. This interaction mode can be carried over to music synthesizers: The user sets as few or as many parameters as he or she likes. Then the system searches the patch database for sounds with corresponding values of these parameters. The synthesizer is set to the patch forming the closest match. If the user is not satisfied with the result, he or she can also retrieve the next best matches or simply adjust the parameters, the ones set before as well as additional ones, and again ask for the best match.

Patch autocompletion presents one major issue as compared to word autocompletion: Only rarely will the matches be exact, since most of the parameters are continuous. Thus, the method also shares some aspects with a text spell-checker, which looks for close but not exact matches. To implement this, one needs a method to measure the "distance" between two patches. In principle, one could base such a distance function on psychoacoustic measurements, for instance through dividing the parameter space into just noticeable differences. However, the parameter space of a standard synthesizer may easily comprise one hundred dimensions; it is hard to see how this could be exhausted by experiments with human listeners. A different option could be to set up a computational model of timbre perception that inputs audio files; this could be used to evaluate the parameter space fully automatically. Such an approach would still face the curse of dimensionality. In addition, psychoacoustic timbre models are still in their infancy [3]. Thus, this work resorts to the data that are at hand: the patch statistics. Based on these data, one can, for instance, determine which values are more probable (see Figure 2) and define a per-parameter distance based on the rank statistics.

To determine a data-based distance between a certain parameter's values 0.4 and 0.8, we count for which percentage of the patches this parameter is larger than or equal to 0.4 but smaller than 0.8. This distance measure is more sensitive at points where the values cluster. For instance, the detune values cluster around zero, so that the distance measure feels a slight detuning as strongly as a strong tuning difference far away from zero.

One can argue whether this approach is reasonable for nominal, discrete-valued parameters, too. For instance, a control for a wave form may offer sawtooth, rectangle, triangle, and sine. Then, the parameter distance between the latter two is the same as between the rectangle and triangle wave, which does not correspond to the perceived degree of similarity. In many instances, however, the assignment of discrete parameters to switch positions conforms to perception. This is true for a frequency range switch with the settings 16", 8", 4", 2" as well as for a filter type switch with the settings LPF, BPF, HPF. Thus, this work sticks to the same simple definition of per-parameter distance for all types of parameters even though this may not be the optimum choice in all situations.

Conflicts in rank order occur when the values of a certain parameter are exactly identical in two patches. To resolve this, every parameter value is shifted by a small random number that is negligible concerning the actual sound.

To find best-matching patches in the library, the overall distance between the settings made by the user and a patch from the library has to be computed. To this end, the per-parameter distances for the parameters set by the user are combined in a Euclidean manner: The total distance is the square root of the sum of the squares of the per-parameter distances. This computation ignores that some parameters may be meaningless. For instance, the LFO's frequency does not matter if the amount of LFO modulation is set to zero. Thus, the system relies on the user providing sensible input: He or she should set the LFO modulation amount to zero and provide *no* further LFO adjustments.

The graphical user interface for this interaction mode consists of the original synthesizer panel plus a parallel coordinates plot of the patch collection, see Figure 3. The parallel coordinates plot recommends typical settings: It allows to read off the value distribution of any single parameter and the correlations this parameter may have with its neighbors. This includes, for instance, dependencies between the ADSR parameters of an envelope generator.

Parameters can be set both in the original interface and in the parallel coordinates plot; they show up as red dots. The parameter values of the best or n-th best match in the sound library are indicated by a polyline. In the prototype, the parameters set by the user retain the precise values the user has set them to and do *not* reflect the values belonging to the patch retrieved from the library. This facilitates tweaking the sound: Small edits still lead to the same nearest patch. On the other hand, however, this approach means that no sound from the library will be reproduced perfectly.

4. CO-VARIATION OF PARAMETERS

The mathematical construct of mutual information between two random variables allows us to learn which parameters bear any kind of statistical relation. In contrast to the usual measurement, which is Pearson's correlation coefficient, mutual information deals well with nominal, discrete-valued parameters such as waveform settings. In addition, it also detects relations such as $y = x^2$ for $x \in [-1, 1]$. Mutual information is measured in bits, giving the typical gain in knowledge one gets about one of the two random variables from knowing the value of the other. If there is no statistical relation, this gain is zero bits.

This work considers only the statistical relations between *pairs* of parameters. In principle, one could also try to infer the best value for a parameter (such as the filter's cutoff frequency) given two others (such as one oscillator's waveform and the amplitude envelope's sustain level). But this faces another curse of dimensionality: There may not be enough patches in the library with this waveform setting and a similar sustain level to warrant a meaningful decision.

Mutual information is not the only mathematical tool to measure the degree of dependency between two random variables. Another prominent choice for this task is conditional entropy. It is not symmetric, however, which makes it hard to visualize, because the *geometric* distance from A to B always equals the distance from B to A.

Depending on the overall settings, some parameters may have little or no influence on the sound. For instance, the speed of an LFO is of less interest when the modulation amount is low. This degree of relevance is taken care of by weights introduced into the computation of the mutual information. These weights stem from a simple model created

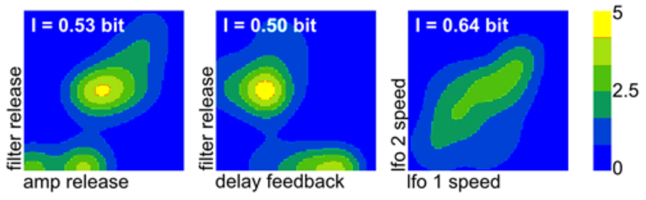

Figure 4: The joint probability density of all pairs of parameters can be determined from the patches in the database (Parzen window of width 0.1).

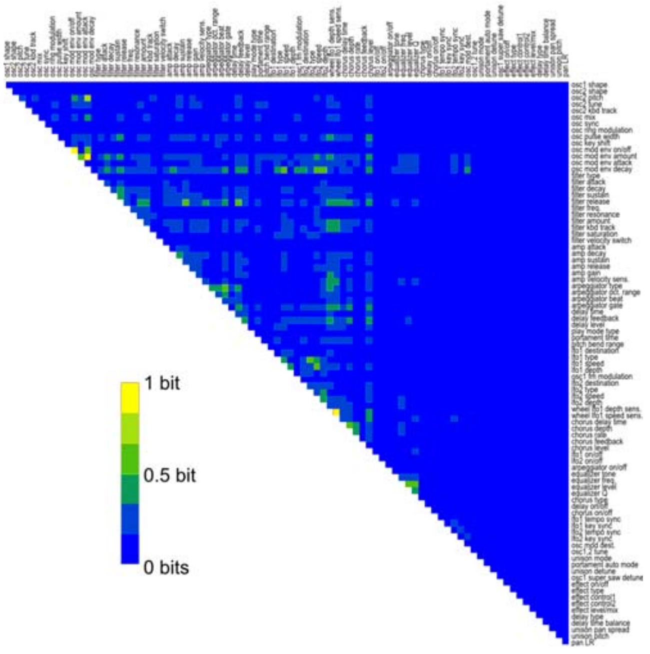

Figure 5: A substantial number of the parameters exhibits clear statistical dependencies in terms of mutual information.

specifically for the current synthesizer. For instance, it assigns a weight w of zero to one to the LFO speed depending on the value of its modulation amount setting.

The weighted mutual entropy $I(X;Y)$ thus amounts to

$$I(X;Y) = \sum_x \sum_y w(x)\, w(y)\, p(x,y) \log_2 \left(\frac{p(x,y)}{p(x)p(y)} \right),$$

where the x are the values of the first parameter X, $p(x)$ is the probability of the first parameter being x, similarly for the second parameter Y, and $p(x, y)$ is the probability of the first parameter being x and the second being y at the same time. Since most of a synthesizer's parameters are continuous, we divide every parameter's range into 16 bins to compute probabilities. This method is a basic approach to estimate probability distributions and needs to be replaced by more sophisticated estimators if the number of samples (that is: sound programs) is low. Figure 4 shows the joint distribution for several pairs of parameters with high mutual information, based on the 1250 patches from the sound library. Figure 5 displays all pair-wise results.

On startup, the prototype software computes the mutual information between all pairs of parameters. It creates a 2D layout of the parameter set that visualizes the statistical relationship through nearness or distance, see Figure 6. To this end, a force-directed graph-layout algorithm [6] is applied that attempts to minimize the en-

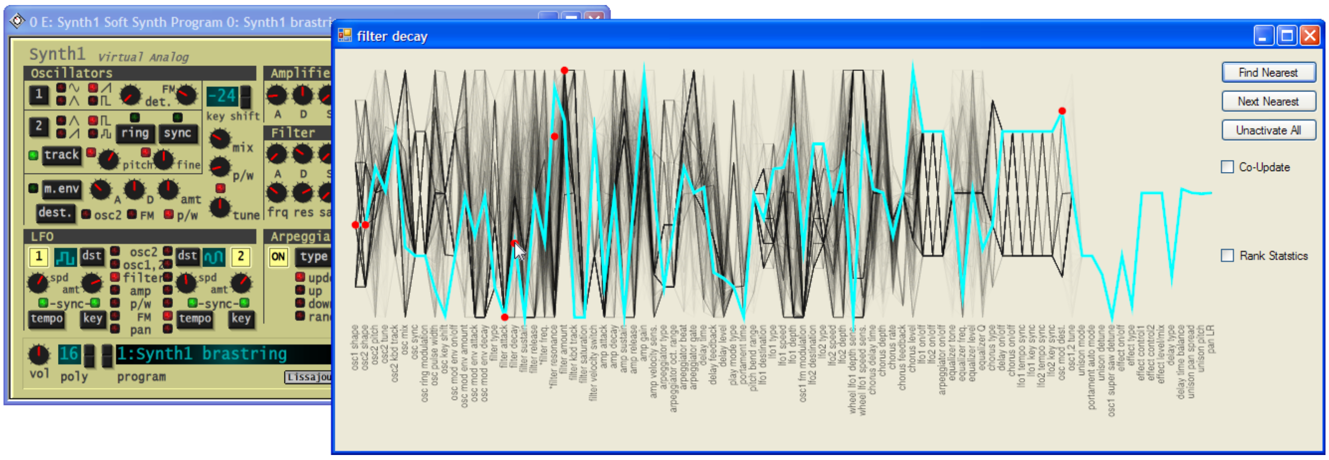

Figure 3: The "autocompletion" interaction mode augments the synthesizer's interface by a parallel coordinates plot of the library.

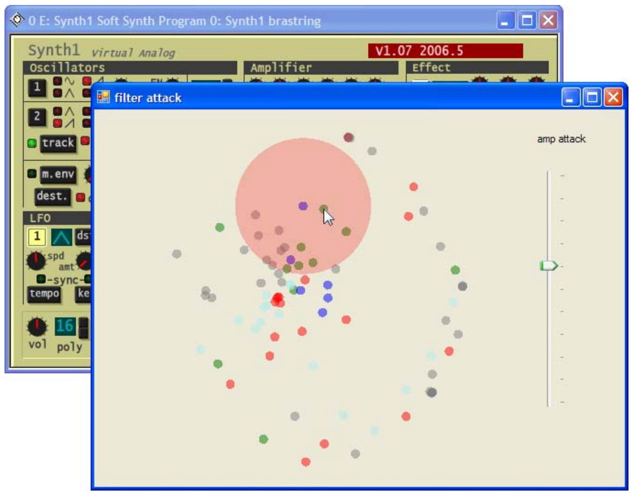

Figure 6: The parameters (represented by dots) are arranged according to their statistical relation, with their colors representing functional groups. The disk indicates the influence radius. The window title names the parameter below the cursor.

ergy $E = \sum_{X \neq Y} \left(1 - d^{\text{actual}}_{X;Y} / d^{\text{target}}_{X;Y}\right)^2$ while staying in a square of 400×400 pixels. Here, $d^{\text{actual}}_{X;Y}$ denotes the distance of the markers representing the parameters X and Y on the screen, and the targeted distance is given by $d^{\text{target}}_{X;Y} = \left(I(X;Y)^3 + \frac{1}{200}\right)^{-1}$, so that unrelated parameters are pushed 200 pixel apart. The third power lets related parameters exhibit a strong pull on each other.

The user can specify an influence radius in this 2D representation to control how many other parameters a change in one parameter will affect. The new value of each influenced parameter is computed through a weighted average of its value in every patch. The relative weight of a patch is $\exp\left(-r^2/(2 \cdot 0.01^2)\right)$, where r denotes the difference of the parameter value set by the user and its value in the patch.

5. CONCLUSION AND OUTLOOK

This work presented two interaction modes that give new meaning to the classic controls of a synthesizer, no matter if they are actual knobs or if they are drawn on a computer's screen. This allows sticking to existing hardware or to existing screen interfaces. Reusing the standard knobs and switches also presents some issues, however. For instance, the standard user interface does not reveal which controls have been set and which have not. On the screen, this could be solved through a semitransparent overlay.

The presented approach may only be the first step toward a statistical evaluation of sound libraries: Can one correlate three or more parameters, possibly through dimensional reduction? [2] Can one create a perception-oriented layout of the parameter controls on the screen? What is the appropriate weighting for sound parameters when computing the "distance" between patches: Is the filter frequency more important than the LFO speed? How can one improve the statistical analysis of the sound library with—manageable—psychoacoustic tests?

6. REFERENCES

[1] R. Bencina. The metasurface: applying natural neighbour interpolation to two-to-many mapping. In *NIME '05*, pages 101–104, 2005.

[2] C. J. C. Burges. Geometric methods for feature extraction and dimensional reduction. In *Data Mining and Knowledge Discovery Handbook*, pages 59–91. Springer, 2005.

[3] J. A. Burgoyne and S. McAdams. Non-linear scaling techniques for uncovering the perceptual dimensions of timbre. In *ICMC 2007*, pages 73–76, 2007.

[4] P. Dahlstedt. A MutaSynth in parameter space: interactive composition through evolution. *Org. Sound*, 6(2):121–124, 2001.

[5] D. P. Ellis. Extracting information from music audio. *Commun. ACM*, 49(8):32–37, 2006.

[6] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 6:24–43, 2000.

[7] M. Hoffman and P. R. Cook. Real-time feature-based synthesis for live musical performance. In *NIME '07*, pages 309–312, 2007.

[8] C. G. Johnson and A. Gounaropoulos. Timbre interfaces using adjectives and adverbs. In *NIME '06*, pages 101–102, 2006.

[9] J. Mandelis and P. Husbands. Don't just play it, grow it!: breeding sound synthesis and performance mappings. In *NIME '04*, pages 47–50, 2004.