SYNTHBOT: AN UNSUPERVISED SOFTWARE SYNTHESIZER PROGRAMMER

Matthew Yee-King
Informatics
University of Sussex

Martin Roth
mhroth@gmail.com

ABSTRACT

This work presents a software synthesizer programmer, *SynthBot*, which is able to automatically find the settings necessary to produce a sound similar to a given target. As modern synthesizers become more capable and the underlying synthesis architectures more obscure, the task of programming them to produce a desired sound becomes more time consuming and complex. SynthBot is presented as an automated solution to this problem. A stochastic search algorithm, in this case a genetic algorithm, is used to find the parameters which produce the most similar sound to the target. Similarity is measured by the sum squared error between the Mel Frequency Cepstrum Coefficients (MFCCs) of the target and candidate sounds.

The system is evaluated technically to establish its ability to effectively search the space of possible parameter settings. A pilot study is then described where musicians compete with SynthBot to see who is the most competent synthesizer programmer, where each competitor rates the other using their own metrics of sound similarity. The outcome of these tests suggest that the system is an effective ”composer's assistant”.

1. INTRODUCTION

As the performance of general purpose computer hardware continues to improve, there is also an associated increase in the complexity of software synthesizers. For example, the current version of Native Instruments' FM8 synthesizer has 1093 parameters [8]. Even with an optimised user interface, creating sounds that are more than mild adjustments of the presets can be a challenge. In this paper we present *SynthBot*, a system which is capable of automatically programming any VSTi compatible software synthesizer in order to produce a sound as close as possible to a target sound supplied by the user.

SynthBot uses a genetic algorithm to search the space of possible parameter settings for any given VST synthesizer plugin [12], guided by a fitness function which compares Mel Frequency Cepstrum Coefficients (MFCC) [5] features of the target sound and the candidate sounds generated by the plugin using these parameter settings. As the candidate parameters evolve, the corresponding synthesized sounds move closer to the target sound and the feature vector error is reduced. When the system finds parameter settings which produce a sound that satisfies the user, this sound can be saved as a preset which is available to any other VSTi host software. The user is able to simply produce completely new programmes for any VSTi synthesizer, tailored to their own specification.

SynthBot is implemented as a cross-platform Java application which uses the Java Native Interface (JNI) to provide a host for the VST plugins and for optimised feature extraction. The system has so far been tested on the Mac OS X and GNU/ Linux platforms. The Java language was chosen to allow rapid cross platform development, especially for GUI and threading functionality.

At the time of writing, there is not a single comparable general purpose, interoperable, and unsupervised synthesizer programmer system available. However, the key techniques - timbre similarity measurement using MFCC features and non-linear parameter optimisation using a genetic algorithm are well established.

In the remainder of this paper, related research is discussed, the technical implementation is described, a technical evaluation is presented along with the initial results of a pilot user evaluation, and finally there is a conclusion and discussion of future plans.

1.1. Related work

There has been a steady interest in the application of unsupervised genetic algorithms to the problem of automatic synthesizer programming. An early example is [7], where tone matching is achieved using FM synthesis and a genetic algorithm. FM synthesis combined with GA programmers appears repeatedly in the literature (e.g. [16, 1, 14], but other synthesis algorithms have been tried, e.g. noise band synthesis [4] and subtractive synthesis [15].

A key question in all these systems is how to judge sound similarity. Most opt for an error measure obtained by comparing the power spectra of the candidate and target sounds, an approach which does indeed reward similar sounds. One problem with the power spectrum is its brittleness - if the same instrument plays two differently pitched notes, there will be a large error between the power spectra even if the notes could clearly be identified by a human user as having been played using the same instrument - human perception must be considered. This is addressed in some of the research, where perceptually informed measures such as the spectral centroid are used (e.g. [16]).

Another key question is how to make these systems usable for musicians. With the exception of [4], the software described in the literature is not publicly available. Where it is available, it does not interoperate with other music software in a way that is appealing to musicians, for example using MIDI or OSC. There is one commercially available system - the Nord Modular G2 patch mutator [10], but this is an interactive GA where the fitness measure is dictated by the user.

In the present work, both of these questions are addressed. Firstly, Mel Frequency Cepstrum Coefficients are used as audio feature vectors, forming the core of the fitness function. The MFCC is considered a suitable measure as it is pitch independent and based on a perceptual model. It is well established in speech recognition [5] and more recently it has been used as a measure of musical similarity [2]. Secondly, a DSP plug-in architecture familiar to many musicians, Steinberg's VST is used to allow our system to access any VSTi compatible synthesizer for the synthesis engine.

The computer music software that has been developed falls in the category of "composer's assistant".

2. IMPLEMENTATION

2.1. Overview

SynthBot is an unsupervised programmer for software synthesizers. It takes as input a target sound file and a software synthesizer, and returns the set of parameters for the synthesizer which produce as similar a sound to the target as possible. MFCCs are used to evaluate sounds similarly to the human ear, and the inverse sum squared error between the target and candidate MFCCs is used to determine the candidate fitness. The application is primarily implemented in the Java programming language, allowing for rapid prototyping and simple GUI development.

The first incarnation of SynthBot works only with VSTi software synthesizers. VSTi plug-ins are written in C++ and compiled natively. They are represented as bundles in Mac OS X, dynamically linked libraries (DLLs) in Windows, and shared objects (SOs) in UNIX/Linux. The SynthBot interface for VSTi plug-ins is thus necessarily also written in C++, as the software synthesizer libraries are only available in that language. A Java Native Interface (JNI) wrapper exposes native VSTi functionality to Java and to the bulk of the SynthBot logic. In this way, SynthBot acts as a VST host, able to control and communicate with VST synthesizers. There are no other known Java-based VST host.

The MFCCs are computed using a custom library called *BoomMFCC*, which is also a native library written in the C programming language and made available to SynthBot via a JNI interface wherein the FFTW library [6] is used to compute discrete fourier transforms. BoomMFCC is optimized for computing MFCCs in multithreaded environments allowing SynthBot to improve performance on modern multicore processors. BoomMFCC was developed following the trial of available MFCC libraries such COMIRVA [11] and LibXtract [3]. The use of BoomMFCC allows increased performance of almost two orders of magnitude over the latter packages. Over the course of development and testing various methods, MFCC computation times were reduced from about 250 milliseconds to roughly 5ms on a modern machine. Performance figures depend heavily on MFCC parameters and implementation details.

2.2. Sound Synthesis

Since the system is compatible with any VSTi plug-in, it is capable of working with any sound synthesis algorithm available in this plugin format. In the evaluations presented here, the freely available mda synthesizer plug-ins *mdaDX10* and *mdaJX10* [9] were used. The mdaDX10 is a single modulator FM synthesizer with 16 paramters and the mdaJX10 is a subtrative synthesizer with 2 tuned and one noise oscillator and 40 parameters.

2.3. Parameter Search

In the VSTi standard, each synthesizer parameter is represented as a real number (a `float`) between zero and one, inclusive. Modern synthesizers may have hundreds of parameters, making the search space high dimensional. Even basic characteristics of search space, such as continuity or variance, are unknown. A stochastic search algorithm is used in order to effectively search for the best parameters. In the present case, this is a genetic algorithm (GA), though others are possible, such as particle swarm optimization (PSO) or simulated annealing (SA).

The GA population begins in a random state. Each individual of the population is represented as an array with length equal to the number of parameters of the synthesizer. Individuals are assessed by loading their parameters into the synthesizer and generating a candidate sound with the same length as the target by passing a fixed MIDI note on message into the synthesizer. The MFCCs of the candidate are computed and the reciprocal of the square distance (sum squared error) to the target MFCCs is used to characterize its fitness. A fitness based proportional roulette wheel as described in [13] is generated and used to select the individuals who will be able to contribute to the next generation. Fitter individuals are more likely to contribute. Crossover occures between each pair of chosen individuals by exchanging subarrays at a uniformly randomly chosen crossover point. Each parameter of the resultant arrays are then mutated by adding a gaussian random variable with zero mean and variance of 0.05. As parameters are constrained to between zero and one, such a mutation will be minor most of the time, and naturally allowing for larger and rarer deviations.

3. EVALUATION

SynthBot has been evaluated in two ways. Firstly, is has been evaluated technically to assess its search behaviour

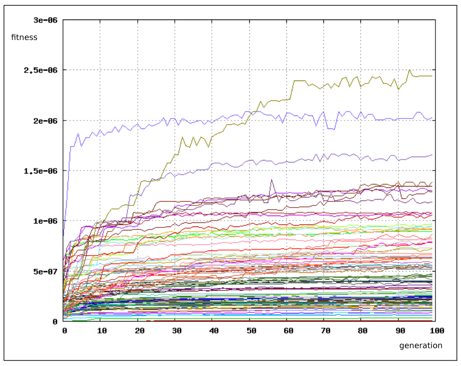


Figure 1. This graph shows fitness against time for 100 runs of the GA.

and performance. Secondly, a pilot study has been carried out to ascertain the usefulness of the system to musicians.

3.1. Technical evaluation

To assess an individual, the plug-in is configured with parameter settings, its output is rendered to a buffer, MFCCs are extracted and the error is calculated. With a window length of 25 milliseconds (ms), an inter-window length of 10ms, 50 mel filter banks and a 1s long sound, this takes around 5 ms on a dual-core 2.2Ghz x86 machine.

In order to establish that the system is capable of effectively searching the space of possible parameter settings, the following process was carried out 100 times:

1. A target audio file is rendered using the mdaJX10 synthesizer with random parameter settings.

2. The optimiser runs for 100 generations with a population size of 100. This takes approximately 3 minutes.

3. Back to 1.

The results of the runs are presented in Figure 3.1. The fitness achieved at the end of the runs has a large variance. This suggests that sometimes a low error is achieved and sometimes a higher error. But what does this mean in terms of the quality of the sound matching? Figure 3.1 shows 4 comparisons between evolved and target specra. The spectra are ranked by the measured MFCC error, where 'a' has the highest error, 'b' has the 33rd highest error, 'c' the 66th and 'd' the lowest error. The variance level indicates that 'b' or 'c' can be expected from a typical run. 'b' and 'c' clearly have spectra which are very close to their targets. Note that the improvement in fitness begins to level off around thirty iterations of the GA, which is only one minute of computation time.

To establish if the system is capable of matching a real instrument timbre is more of a challenge as it was soon observed that the spectra of evolved and target sounds did not look anywhere near as similar as the test sounds from above, even when the perceived similarity was quite compeling.

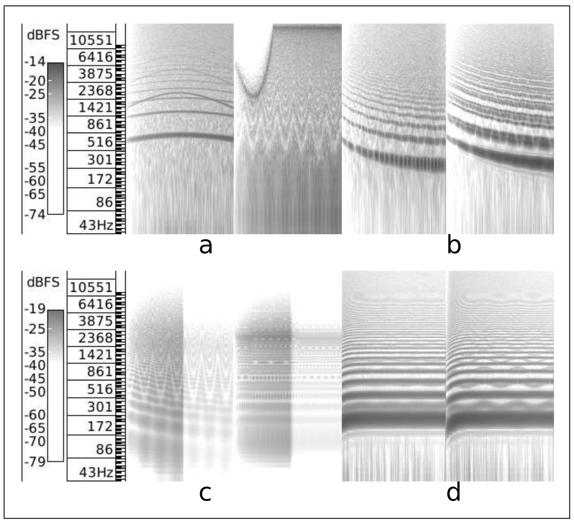


Figure 2. This image shows spectrograms of the worst(a), 66th best(b), 33rd best(c), and best(d) results from the technical evaluation. The target spectrum is shown next to the candidate spectrum for each run.

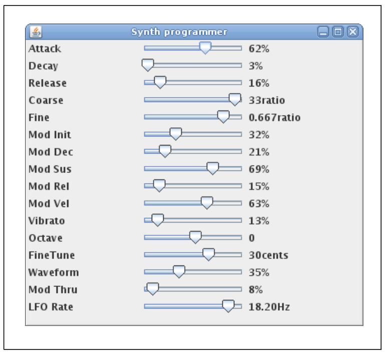
3.2. Experimental evaluation

Objective error measures are not sufficient to rate the usefulness of the system when evolving towards real sounds. A subjective evaluation is conducted using a two phase experiment in which the performance of expert human users is compared to that of SynthBot. In phase one, ten expert human users are asked to programme two sounds on the mdaJX10 and mdaDX10 synthesizers using a generic interface as shown in Figure 3.2. For each synthesizer the targets are a real instrument sound and a pitched sound made using that synthesizer. The real instrument sounds cannot be exactly reproduced using the synthesizers but the synthesized sounds can be. SynthBot is given the same task and this results in a total of 44 sounds. In phase two an online evaluation is carried out where users rate each of the 44 sounds for similarity to their respective target sounds. SynthBot then rates the sounds using its MFCC error metric. Finally correlations are sought between the similarity ratings of the human users and those of SynthBot. This experiment should establish the quality of SynthBot's programming performance as well as the usefulness of the MFCC error metric for instrument sounds. At this stage, phase one is complete and phase two is ongoing.

Initial results indicate that SynthBot achieves an MFCC error which is an order of magnitude smaller than the human, in about half of the time. The qualitative feedback has been encouraging - the subjects would certainly use such a tool, they say, and would like to see how it performs with their own VSTi plug-ins.

4. CONCLUSION

An automatic software synthesizer programmer, *SynthBot*, has been presented. It is capable of loading any VSTi



Synth programmer

Attack	62%
Decay	3%
Release	16%
Coarse	33ratio
Fine	0.667ratio
Mod Init	32%
Mod Dec	21%
Mod Sus	69%
Mod Rel	15%
Mod Vel	63%
Vibrato	13%
Octave	0
FineTune	30cents
Waveform	35%
Mod Thru	8%
LFO Rate	18.20Hz

Figure 3. The interface used to carry out the musician tests, showing the paramters for the mdaDX10 synthesizer

compatible software synthesiser and finding the parameters which cause it to produce a sound closest to a given target. Sound similarity is measured according to squared distance between candidate and target MFCCs, and candidate optimisation is accompished using a genetic algorithm. SynthBot is a composer's assistant intended to save time and effort, as the synthesiser architectures become more complex and the knowledge necessary to program them in a meaningful way becomes more arcane. The system is evaluated according to technical and experimental means. The former shows how the spectrum of the candidate sound evolves to meet that of the target, usually in a matter of minutes. The latter pits SynthBot against musicians and the preliminary results indicate that the program outperforms even experienced synthesiser programmers.

Future Work

There are many directions for future work. This includes improved software synthesiser plugin support, not only for more advanced VSTs, but other common interfaces such as AudioUnits or Linux Audio Developer's Simple Plugin API (LADSPA). Further improvements in target sound acquisition are also necessary, possibly through the use of a better tuned GA or another stochastic search algorithm such as Particle Swarm Optimisation (PSO). Improvements in fitness function are also envisioned. This includes not only optimised computation of MFCCs, but also other better tuned metrics.

Acknowledgments

The authors would like to thank the musicians who were invaluable in conducting the experimental evaluation.

5. REFERENCES

[1] L.J.S.M. Alberts. Sound reproduction using evolutionary methods: A comparison of fm models. Website, 2005.

[2] Franois Aucouturier, Jean-Julien ; Pachet. Tools and architecture for the evaluation of similarity measures : Case study of timbre similarity. In *Proceedings of ISMIR Conference 2004*, 2004.

[3] Jamie Bullock. Libxtract: A lightweight library for audio feature extraction. In *Proceedings of the ICMC 2007 International Computer Music Conference*, 2007.

[4] Michael Chinen and Naotoshi Osaka. Genesynth: Noise band-based genetic algorithm analysis/synthesis framework. In *Proceedings of the ICMC 2007 International Computer Music Conference*, 2007.

[5] Mermelstein P. Davis, S.B. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. In *EEE Trans. on Acoustic, Speech and Signal*, pages 357–366, 1980.

[6] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. special issue on "Program Generation, Optimization, and Platform Adaptation".

[7] A Horner, J Beauchamp, and L Haken. Genetic Algorithms and Their Application to FM, Matching Synthesis. *Computer Music Journal*, 17(4):17–29, 1993.

[8] Native Instruments. Fm8 the power of digital. Website, 2008.

[9] Paul Kellet. mda-vst.com. website, 2008.

[10] Clavia Palle Dahlstedt. Nord modular g2 patch mutator. website, 2006.

[11] Markus Schedl. The CoMIRVA Toolkit for Visualizing Music-Related Data. Technical report, Department of Computational Perception, Johannes Kepler University Linz, June 2006.

[12] Steinberg. Vst audio plug-ins sdk 2.4 revision 1. website, 2008.

[13] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[14] Matthew John Yee-King. An automated music improviser using a genetic algorithm driven synthesis engine. In *EvoWorkshops*, volume 4448 of *Lecture Notes in Computer Science*, pages 567–576. Springer, 2007.

[15] Matthew John Yee-King. The evolving drum machine. Music-AL workshop, ECAL conference 2007, 2007.

[16] Der-Tzung Liu Yuyo Lai, Shyh-Kang Jeng and Yo-Chung Liu. Automated optimization of parameters for fm sound synthesis with genetic algorithms. In *2006 International Workshop on Computer Music and Audio Technology*, 2006.