SOUND MATCHING USING SYNTHESIZER ENSEMBLES

Gerard Roma

School of Computing and Engineering
University of West London
London, UK
gerard.roma@uwl.ac.uk

ABSTRACT

Sound matching allows users to automatically approximate existing sounds using a synthesizer. Previous work has mostly focused on algorithms for automatically programming an existing synthesizer. This paper proposes a system for selecting between different synthesizer designs, each one with a corresponding automatic programmer. An implementation that allows designing ensembles based on a template is demonstrated. Several experiments are presented using a simple subtractive synthesis design. Using an ensemble of synthesizer-programmer pairs is shown to provide better matching than a single programmer trained for an equivalent integrated synthesizer. Scaling to hundreds of synthesizers is shown to improve match quality.

1. INTRODUCTION

Software synthesizers are widely used in computer-based music production. In this context, *sound matching* can be described as a mode of interaction with a software synthesizer where the user provides an example of a target sound, and the system automatically provides a set of parameters that will make the synthesizer approximate the target. Real-world interaction with sound synthesizers is rarely this simple: it can be seen as a two-way process where the user may have some goal but can also discover interesting sounds in the process by tweaking parameters, even if blindly. Regardless, given the complexity of some synthesizers, sound matching can help find configurations that would otherwise be missed.

Sound matching is an application of *automatic synthesizer programming*, which more generally describes software systems able to automatically find synthesizer parameters. The origins of sound matching can be traced back to the research on the synthesis of musical instrument sounds using frequency modulation (FM) synthesis [1], which is notoriously hard to program.

After the development of FM, research on sound synthesis mostly focused on computational models, such as physical models and spectral models, that would make it easier to reproduce and control known sounds [2]. Spectral modelling allowed automating additive synthesis using spectral analysis, which could eventually replace the need for sound matching techniques, however, it has mostly been applied to the transformation of digital sound samples. Along with the use of samples, software synthesizers based on classic techniques, such as wavetable synthesis, FM or digital simulation of subtractive synthesis (AKA analogue modelling) remain popular in music production.

In recent years, sound matching has continued to gain attention in music technology research. Most recent approaches have focused on software-based FM synthesizers [3, 4] or otherwise commercially available plug-ins that are treated as a black box.

This paper proposes a different approach to sound matching. Instead of finding how to program an existing synthesizer, the focus is on how to reproduce the sound presented by the user. In sound matching research, this is often known as matching *out-of-domain* sounds in the sense that the target sound may not have been originally produced by a particular synthesizer. One prominent incentive for matching an arbitrary sample with a synthesizer is that a synthesizer allows further tweaking (even if blindly) and obtaining many possible variations of the sound, which can be more useful for music and audio production than a single sample. However, since a given synthesizer cannot reproduce all possible sounds, having more than one synthesizer could increase the possibilities for matching arbitrary samples. This paper presents an initial study for such ensemble-based sound matching. Each synthesizer configuration is used to train a neural network which is used to map an existing sound to a set of synthesizer parameters. An ensemble of paired synthesizers and programmers is then used to match the target sound.

The paper is organized as follows: the next section reviews existing literature on sound matching. Section 3 describes the proposed approach based on synthesizer ensembles. In Section 4, an implementation using SuperCollider is described. The proposed approach is validated through several experiments in Section 5, which are discussed in Section 6. Section 7 outlines future work.

2. RELATED WORK

Synthesizer programming techniques can be divided into *search-based* and *modelling-based* [3]. In the first case, the algorithm tries to find a suitable parameter set for a target sound by searching the space of sounds of a synthesizer. This is typically done using genetic algorithms (GA) [1, 3, 5, 6, 7]. Search-based approaches are inherently limited for interactive applications due to the time it takes to assess each candidate as part of the search process. In the case of real-time synthesis, the generation of each candidate takes the same time as its duration, so in all the search typically lasts at least several minutes.

Modelling-based approaches typically consist of pre-trained models, such as neural networks [3, 4, 8]. In this case, the model is trained to learn a mapping between a synthesized sound and the set of parameters that the synthesizer used to generate it. Most works focus on the parameter space of existing software synthesizers available as plug-ins. In many cases, the training sets are collected from real-world preset databases. Systems based on existing synthesizers are also limited in that the synthesizer programmer is focused on the space of sounds produced by a particular synthesizer and synthesis technique. In the literature, these are typically referred to as *in-domain* sounds, whereas *out-of-domain* sounds are sounds produced by other means [5]. Focusing on existing synthesizers, significant progress has been made by using generative modelling, which allows interpolation of presets and navigation of the parameter space based on perceptual dimensions [9, 10].

One important problem of modelling-based approaches is that optimization is based on a loss function evaluated on the parameter space. In other words, the system tries to learn the mapping from sound to parameters by minimizing the difference between the predicted parameters and the real parameters of training examples. However, the distance in parameters is not the same as the perceptual difference between sounds. Some works have explored using differentiable DSP (DDSP) [11] to overcome this limitation, by designing synthesizers that can be incorporated in the neural network model. This allows models to directly optimize the spectral distance between the target and the predicted sound [12, 13]. In a way, DDSP can be seen as a deep-learning version of spectral modelling synthesis [14]. Thus, these systems tend to learn the temporal evolution of parameters, such as the frequencies of oscillators, which are implemented in hybrid additive-subtractive synthesizers. In addition, the synthesizers are implemented using differentiable machine learning routines. This means that at the moment practical applications are limited since such synthesizers do not correspond to commonly available implementations.

Given that the range of sounds that each synthesizer can produce is limited, some works have looked at matching sounds using more than one synthesizer design. The system in [6] used genetic programming to evolve different signal processing graphs, implemented as Pure Data patches. More recently, the VAE-based approach in [9] has been extended to more than one synthesizer by using multiple decoders targeting several commercial synthesizers from a shared latent space [15]. The present work similarly explores matching with multiple synthesizers but following a different approach. Rather than evolving patches using genetic programming, a repertoire of designs is generated beforehand. Then for each design, a separate programmer model is trained. There is no shared latent space, although a global classifier is used to choose the most suitable synthesizer and programmer.

3. MATCHING WITH SYNTHESIZER ENSEMBLES

A synthesizer can be seen as a patch that connects different modules, such as oscillators, envelope generators and filters. Given the infinite possibilities offered by modular synthesis, designers of classic analogue synthesizers created more limited configurations that allowed switching between a few module types. Software synthesizers could similarly offer infinite routing possibilities, as do Music-N-like computer music systems such as Max, Pure Data or SuperCollider [16]. However, popular software synthesizers usually replicate the analogue style and offer a limited set of configurations. While many different synthesis techniques are available in the digital domain, analog modelling (i.e. simulation of subtractive synthesis) remains a popular and well-understood paradigm.

For the purpose of this study, a 'synthesizer' is defined as a specific patch with specific modules. Given a fixed structure, many possible synthesizers can be defined. Thus, synthesizer ensembles can be generated using structural templates. As an example, the template used for this study is shown in Figure 1. Here, oscillators could have different waveforms, including also white noise, and modulators could be low-frequency oscillators (also with different waveforms) or envelope generators. Different types of filters (e.g. low-pass, high-pass) can also be used. The 'amplifier' is just a gain control that allows using a modulator to control the amplitude. A particular choice of oscillators, modulators and filters will instantiate the template into a specific synthesizer. Each synthesizer will produce a different range of sounds and also offer a different set of parameters depending on the choice of modules. This is unlike common commercial synthesizers where different patches may be available in the same interface. In that case, some parameters, such as the choice of oscillator, will be discrete and result in significantly different sounds.

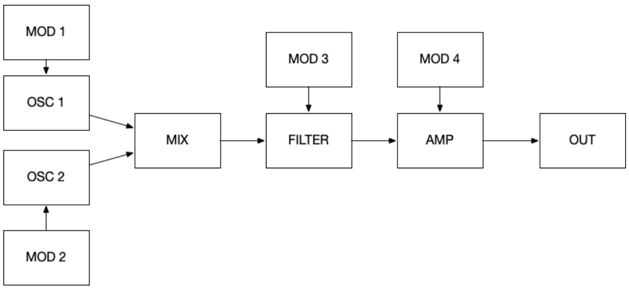

Figure 1: Subtractive synthesizer template.

While the example in Figure 1 is a common subtractive synthesizer configuration, many more are obviously possible, including different types of synthesis. An 'ensemble' can thus be defined as a number of concrete synthesizers, each with a set of parameters and a sound space. An ensemble can be produced by a single or multiple templates, and thus ensembles could be created by combining a variety of synthesis techniques.

For the purpose of sound matching, a synthesizer can be described as a function that maps a set of parameters to an audio excerpt. Conversely, a synthesizer programmer maps from an audio excerpt (often parametrized as a feature matrix) to a set of parameters. In the case of a synthesizer ensemble, each synthesizer is associated with a specific programmer and vice versa. In this study, programmers are implemented as neural network regression models, which are trained using random samples of the synthesizer parameter space.

Given a target audio sample and an ensemble of synthesizer-programmer pairs, the problem is then selecting a suitable pair from the ensemble. In this paper three approaches are investigated:

- **Select Best**: The best synthesizer is selected by evaluating all the programmers with the target sound, synthesizing a new sound with the resulting parameters, and comparing it with the target sound. The model that produces the sound closest to the target is selected. This can be seen as a brute-force approach, as all the programmers need to be evaluated and all results compared with the target.
- **Predict Same**: A classifier is trained with the same data as the programmers, and learns to predict which synthesizer generated which sample. The target sample is assumed to have been generated by one of the synthesizers (which is not the case for out-of-domain samples). The selected synthesizer is then the one predicted to have generated the target.
- **Predict Best**: For predicting with out-of-domain sounds, this approach uses an extra dataset of out-of-domain sounds,

which has not been used for training. For each sound, the best-performing synthesizer-programmer is chosen as per the Select Best approach. A classifier is then trained to predict which synthesizer would produce the best match for the target sound, regardless of whether it has been generated by one of the synthesiers.

It can be expected that, given simple synthesizers such as the ones generated by the subtractive synthesis template described above, the larger the ensemble the better the possibilities to obtain a sound similar to the target.

4. IMPLEMENTATION

In order to test the proposed approach, a system for automatic generation of synthesizers from a given template was implemented using the SuperCollider language [17] (the software, along with the code for experiments, can be accessed at `https://github.com/g-roma/SoundMatchEnsemble`). Each of the modules in Figure 1 can be implemented using a number of unit generators (UGens).

Since the language offers a large number of UGens, this system could result in many possible synthesizers. For the purpose of this study, the choice was restricted to UGens shown in Table 1. Here, LFTri is a triangle LFO, VarSaw is a Sawtooth oscillator with a variable duty cycle, and DC is a constant number (no modulator). The filter can be either a low-pass, band-pass or a high-pass resonant filter. Following previous work (e.g. [3, 4]) the pitch is fixed to a single note for all synthesizers (C4 note). Each UGen / module offers different parameters, so each particular synthesizer will potentially have a different number of parameters. The parameters are shown in Table 2. For this template, *OSC 1* is an oscillator with variable pulse width control, which is modulated by *MOD 1*. *MOD 2* controls either the frequency (Saw) or the amplitude (WhiteNoise) of *OSC 2*. The envelope for EnvGen is always specified as an ADSR envelope.

Module	UGens
OSC 1	Pulse, VarSaw
OSC 2	Saw, WhiteNoise
MOD 1-4	LFTri, EnvGen, DC
FILTER	RLPF, RHPF, BPF

Table 1: UGens used for template-based generation

UGen	Parameters
Pulse, VarSaw	Modulation amount, Pulse width
Saw, Noise	Modulation amount
LFTri	Frequency
EnvGen	Attack, Decay, Sustain, Release
RLPF, RHPF, BPF	Cutoff, Modulation amount, 1 / Q
Mixer	Source 1 / 2 balance

Table 2: Parameters for each UGen and module

Each synthesizer is identified by a 7-digit string where each digit indicates the UGen used for each module of the template. For example, the synthesizer with ID 1120010 is shown in Figure 2, along with all the resulting parameters.

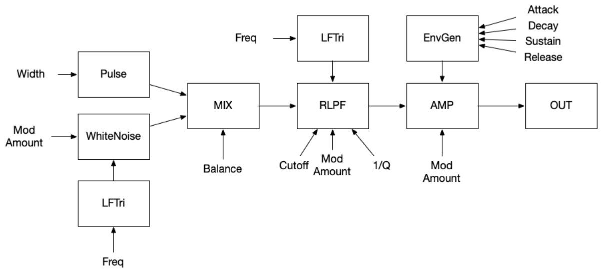

Figure 2: Synthesizer instance with 13 parameters

Synthesizer instances are controlled by arrays of parameters normalized to the 0-1 range. In the software framework, each instance is an object that can generate a random set of parameters (i.e. it knows the specific number of parameters for this synth), and also produce a sound given an input set of parameters. Parameters are internally mapped to pre-defined ranges and curves for each UGen.

Programmers and classifiers are implemented in Python using the Pytorch library [18]. For training each programmer, a set of random parameter sets are generated with a synthesizer instance, and the corresponding sounds are recorded to audio files and converted to a matrix of Mel Frequency Cepstral Coefficients (MFCC) using librosa [19].

5. EXPERIMENTS

The proposed approach was validated through three experiments. First, different network architectures were compared as individual synthesizer programmers. A second experiment compared a 'monolithic' synthesizer incorporating module choices as parameters with an equivalent ensemble of 12 simple synthesizers. Finally, a third experiment investigated scaling to larger ensembles.

In all experiments, synthesizer instances were used to generate 4-second recordings, with envelopes sustained for 3 seconds, at a 48Khz sample rate. MFCCs were obtained with the default librosa settings, which generated matrices of 20 coefficients by 376 frames (each frame was generated by a 10ms hop).

Target and predicted audio samples were compared using the MFCC distance used in [3, 4], defined as:

$$MFCCD(S,T) = \frac{1}{C}\sum_{c}\sqrt{\frac{1}{N}\sum_{n}(S_{c,n}-T_{c,n})^2}, \quad (1)$$

where C (indexed by c) is the number of MFCC coefficients, and N (indexed by n) is the number of frames. S and T are source and target MFCC matrices. This metric has been found to be easy to interpret, with values around 10-15 or less considered to be close matches [3, 4].

Training used 5000 examples for each network with 10% used for validation and 10% for testing. Validation was used for early stopping. All models were trained for a maximum of 100 epochs. Examples were obtained by random sampling of the parameters of the corresponding synthesizer. Training was performed by an Adam optimizer [20] using mean square error (MSE) loss over the parameters.

5.1. Individual programmers

Several neural network architectures were compared for learning the mapping from MFCC to synthesizer parameters for an individual synthesizer. The experiment was run for two synthesizers: one relatively simple (13 parameters, ID 1022210) and one more complex (24 parameters, ID 1111111). The following architectures were compared:

- Multi-Layer Perceptron (MLP) with LeakyRELU activations
- Convolutional neural network (CNN) with LeakyRELU activations
- Bi-directional Long short-term memory network (BiLSTM)

All networks were configured empirically to maximize performance while keeping comparable training times. Performance was measured by the average parameters MSE distance and the average MFCC distance on the test set (in-domain sounds). The results were averaged over 10 runs.

Results are shown in Table 3, including the time taken to train each model for 5000 examples on an Apple M2 Pro GPU. Preliminary experiments showed that both MLP and CNN models could obtain good results for the (more difficult) complex synthesizer by increasing the capacity while using early stopping as regularization. For the MLP this resulted in worse performance for the basic synthesizer, while the CNN model showed overall good performance and shorter training times, so it was selected for subsequent experiments. An analysis of the results for the 24-parameter synthesizer showed that the resulting sounds typically replicated the spectral content of the target, but often failed to precisely reproduce the amplitude envelope. An example match is shown in Figure 3. This synthesizer included a mixture of a pulse oscillator and white noise. The automatic programmer was able to replicate the mixture of both and part of the attack and release behaviours but failed to reproduce all the nuances of the amplitude envelope. For randomly generated in-domain sounds, the MFCC distance was a good predictor of match quality, but there were still noticeable differences for values below 10. This can be attributed partly to the randomness of the dataset, as there is no sound design that can guide the perception.

5.2. Monolithic vs ensemble

A second experiment compared a ‘monolithic synthesizer’ with a small equivalent ensemble. The monolithic synthesizer was generated similarly to the method described in Section 3, but with some parameters that could switch between different modules (OSC 1 and 2, FILTER). The rest of the modules were set to predefined values (LFTri for MOD 1, DC for MOD 2, EnvGen for MOD 3 and 4). The choice was made so that the number of parameters was fixed in any case, giving a total of 19. Parameters for choosing the oscillators and filter were normalized from 0 to 1 and internally quantized. The monolithic synthesizer was compared to the equivalent ensemble of 12 synthesizers, one for each configuration (2 oscillators × 2 oscillators × 3 filters).

For choosing the ensemble synthesizer, the methods described in Section 3 were compared. Evaluating all models and picking the best result (*Select Best*), using a classifier trained with in-domain samples (*Predict Same*) or using a classifier trained with out-of-domain samples (*Predict Best*). For the classifier-based methods, the CNN model from Section 5.1 was repurposed as a classifier and trained with the ensemble examples. The classifier achieved a multi-class accuracy of 85% for *Predict same* and 69% for *Predict best*.

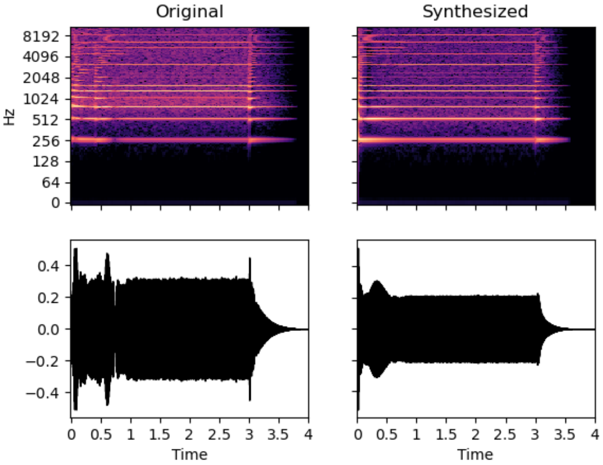

Figure 3: Example of in-domain match

The different methods were compared using MFCC distance with out-of-domain samples. The samples were obtained from the NSynth dataset [21], selecting all the samples with the same pitch used in the synthesizers (C4). The training subset of this dataset (4000 samples for C4) was used to train the classifier for the *Predict Best* approach, whereas the validation set (174 samples for the same note) was used to evaluate al the methods.

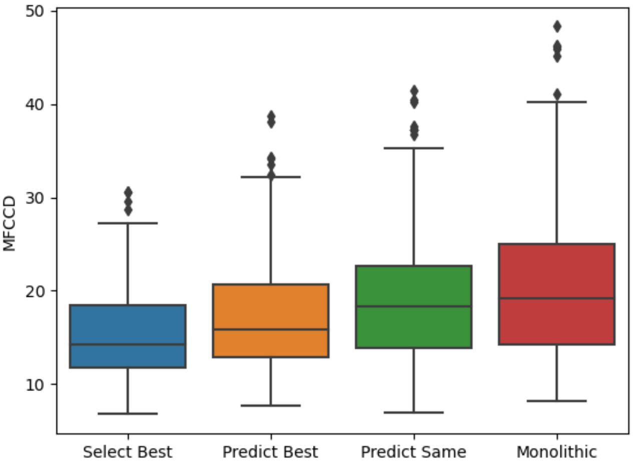

Figure 4: MFCCD distributions for out-of-domain samples

Figure 4 shows the distributions of the MFCCD metric for all the methods. Boxes show the quartiles of the data, while whiskers show the whole distribution range except for outliers. All groups were shown to be significantly different through a Wilcoxon signed-rank test ($p < 0.01$). The results show that the ensemble method

	13 parameters synth			24 parameters synth		
Network	MSE (param)	MFCCD	Training time (s)	MSE (param)	MFCCD	Training time (s)
MLP	0.1	6.57	231	**0.06**	11.8	265
CNN	**0.05**	**4.3**	**120**	0.07	**11.5**	**118**
BiLSTM	0.07	9.9	295	0.08	16.2	317

Table 3: Comparison of network architectures

can obtain better results for matching out-of domain sounds as compared to an equivalent integrated synthesizer. Following the *Select Best* approach, *Predict Best* was shown to perform significantly better than *Predict Same* approach, which in turn was significantly better than the monolithic synthesizer.

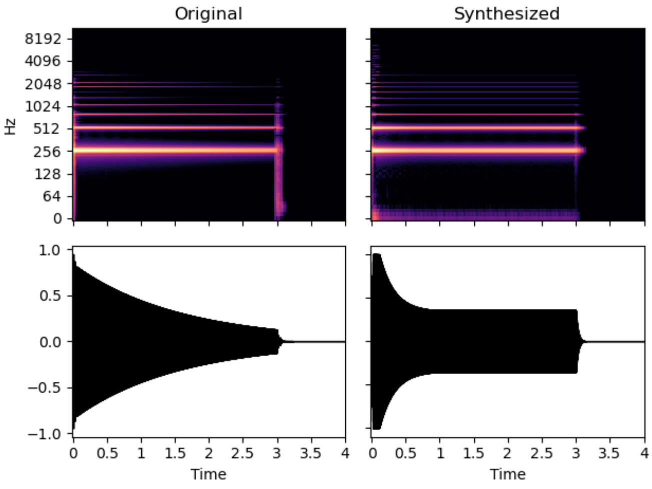

Figure 5: Example of out-of-domain match

5.3. Scaling to larger numbers

For testing the potential for scaling to larger ensembles, all the possible synthesizers for the implementation in Section 4 were generated, using always an envelope for amplitude modulation. This resulted in 324 synthesizer - programmer pairs. Ensembles of increasing sizes were sampled randomly to test the effect of the ensemble size, using the *Select Best* method with the same out-of-domain samples as in the previous experiment. Evaluation of other selection methods for larger scales was left for further work. Results are shown in Figure 6. Error bars show 95% confidence intervals for the distribution of MFCC distance. The distance decreases monotonically, which supports the hypothesis that larger numbers of synthesizers can help obtain better matches, while obviously increasing the computational cost. Figure 7 shows the distribution of MFCC distance when using the full ensemble of 324 synthesizers. Most of the examples (79%) were matched below a value of 15.

For out-of-domain samples, the distance was subjectively found to correlate better with perception than with in-domain sounds. However, the quality of the match strongly depended on how feasible it was for the simple subtractive synthesis design to replicate the acoustic properties fo the out-of-domain sound. An example is shown in Figure 5. Here, an acoustic guitar sound was matched with the subtractive synthesizer. Like in the case of in-domain sounds, the synthesizer was able to replicate the harmonic structure, and some of the amplitude envelope. Yet, since the ADSR controls were designed to be independent and add up to 4 seconds (e.g. the maximum decay time was 1 second), the synthesizer could not perfectly replicate the acoustic decay envelope.

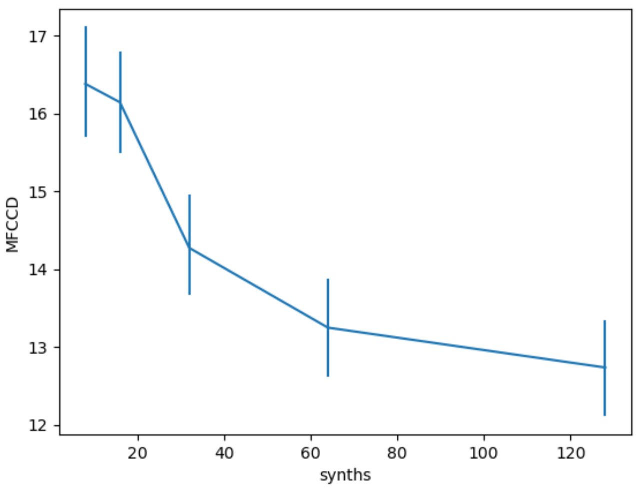

Figure 6: MFCCD distributions for out-of-domain samples for ensembles of increasing size

6. DISCUSSION

The experiments show that good results, in terms of MFCC distance, can be obtained with simple synthesizer designs both for in-domain and out-of-domain samples. Used as ensembles, these designs can perform better than equivalent integrated designs for sound matching. From subjective listening, the MFCC distance proved to be a useful metric for assessing the quality of matches. However, for the single-note sounds in the NSynth dataset, this distance mostly emphasized similarities in spectral content, while errors in the envelope and modulations were often not reflected. This may be due both to the frame-based nature of the distance and the logarithmic amplitude of MFCCs. This is compounded by the difficulty of matching out-of-domain sounds, which may have complex envelopes, with an ADSR envelope generator, as observed also in [13].

With respect to the network architecture, during training, it seemed clear that good results could be obtained with different architectures. Short training times were treated as a priority in order to investigate the use of large ensembles with potentially hundreds of programmers. Using random sampling of parameters allows

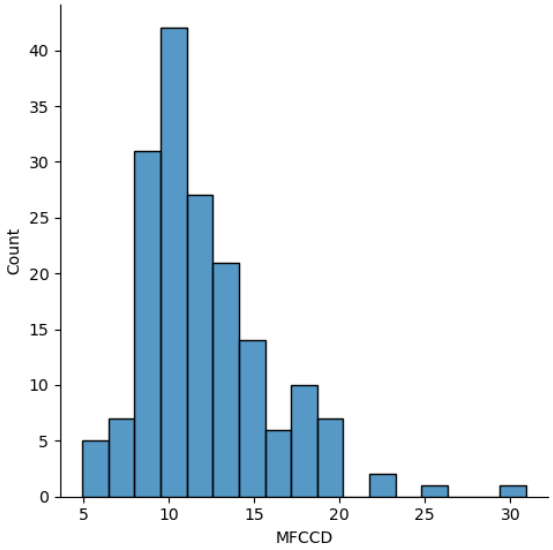

Figure 7: MFCCD distribution for out-of-domain samples using 324 synths ensemble

creating datasets of arbitrary size, which could be used to create better models, while at the same time is probably more challenging than using real-world presets as commonly done in recent work [4, 9, 10].

Concerning ensemble-based matching, the best results were obtained by evaluating all the programmers in the ensemble and selecting best-matching sample from the ensemble. This approach could be used in practical applications but would require more time for rendering each sound than the classification-based approaches. Unlike in GA-based approaches, where samples are generated at each iteration, here they only need to be generated once, which can be done in parallel for real-time synthesis of many examples. However, for larger ensembles, this approach could become a bottleneck.

The proposed template-based system can be used flexibly to generate many kinds of synthesizers. A general limitation of the ensemble approach is that, given a target sound, the user would be presented with a potentially different interface each time. Given that the goal is to be able to modify the sounds recreated by the synthesizer, this assumes a certain familiarity with synthesis concepts, such as subtractive synthesis modules, rather than an expertise with a given synthesizer. This is an opportunity for designing novel user interfaces where some elements are retained and some are varied.

Beyond the limitations of the system itself, the study is limited in different ways. Compared with targeting commercial synthesizers, the proposed system requires many design decisions. For example, early versions of the system used with random parameter sampling tended to produce many more modulations than is common in real-world usage. Also, random modulators had to be discarded, as they would produce a different sound each time, which impaired the evaluation based on MFCC distance.

7. CONCLUSIONS AND FUTURE WORK

This paper has presented a method for sound matching using custom synthesizers generated from templates. The system is implemented using SuperCollider and Python, and is available as open-source software. The experiments have shown the potential for sound matching using the NSynth dataset.

Given the possibilities offered by template-based generation, this study has only scratched the surface. More work is needed in a number of directions.

First, the study has only evaluated the different classification-based selection techniques for a small ensemble. Therefore, future work will focus on the selection procedure at larger scales.

A second important direction is studying mixtures of different synthesis techniques. This could allow using sound matching with broader ranges of out-of-domain sounds, eventually allowing systems that provide interesting results for virtually any kind of sound.

Finally, the selection of synthesizers with different parameters for the matching use case will likely pose some interesting challenges with respect to the user interface. At the same time, implementing the system in a playable interface could introduce further requirements for selecting the synthesizer (for example focusing on versatility). In general, the usability of the proposed approach should be investigated in a user study.

8. ACKNOWLEDGMENTS

This work has received support from the Learning and Development scheme at University of West London.

9. REFERENCES

[1] Andrew Horner, James Beauchamp, and Lippold Haken, “Machine tongues XVI: Genetic algorithms and their application to fm matching synthesis,” *Computer Music Journal*, vol. 17, no. 4, pp. 17–29, 1993.

[2] Julius O Smith III, “Viewpoints on the history of digital synthesis,” in *Proceedings of the International Computer Music Conference*. ICMA, 1991, pp. 1–1.

[3] Matthew John Yee-King, Leon Fedden, and Mark d’Inverno, “Automatic programming of VST sound synthesizers using deep networks and other techniques,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 150–159, 2018.

[4] Zui Chen, Yansen Jing, Shengcheng Yuan, Yifei Xu, Jian Wu, and Hang Zhao, “Sound2synth: Interpreting sound via fm synthesizer parameters estimation,” *arXiv preprint arXiv:2205.03043*, 2022.

[5] Naotake Masuda and Daisuke Saito, “Quality diversity for synthesizer sound matching,” in *2021 24th International Conference on Digital Audio Effects (DAFx)*. IEEE, 2021, pp. 300–307.

[6] Matthieu Macret and Philippe Pasquier, “Automatic design of sound synthesizers as pure data patches using coevolutionary mixed-typed cartesian genetic programming,” in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 309–316.

[7] Yuyo Lai, Shyh-Kang Jeng, Der-Tzung Liu, and Yo-Chung Liu, "Automated optimization of parameters for FM sound synthesis with genetic algorithms," in *International Workshop on Computer Music and Audio Technology*. Citeseer, 2006, p. 205.

[8] Oren Barkan and David Tsiris, "Deep synthesizer parameter estimation," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 3887–3891.

[9] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, Axel Chemla, et al., "Universal audio synthesizer control with normalizing flows," in *International Conference on Digital Audio Effects (DAFx 2019)*, 2019.

[10] Gwendal Le Vaillant, Thierry Dutoit, and Sébastien Dekeyser, "Improving synthesizer programming from variational autoencoders latent space," in *2021 24th International Conference on Digital Audio Effects (DAFx)*, 2021, pp. 276–283.

[11] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts, "Ddsp: Differentiable digital signal processing," *arXiv preprint arXiv:2001.04643*, 2020.

[12] Naotake Masuda and Daisuke Saito, "Synthesizer sound matching with differentiable dsp.," in *ISMIR*, 2021, pp. 428–434.

[13] Naotake Masuda and Daisuke Saito, "Improving semi-supervised differentiable synthesizer sound matching for practical applications," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 31, pp. 863–875, 2023.

[14] Xavier Serra and Julius Smith, "Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition," *Computer Music Journal*, vol. 14, no. 4, pp. 12–24, 1990.

[15] Daniel Faronbi, Iran Roman, and Juan Pablo Bello, "Exploring approaches to multi-task automatic synthesizer programming," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5.

[16] Victor Lazzarini, "The development of computer music programming systems," *Journal of New Music Research*, vol. 42, no. 1, pp. 97–110, 2013.

[17] James McCartney, "Rethinking the computer music language: Supercollider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, 2002.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[19] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto, "librosa: Audio and music signal analysis in python.," in *SciPy*, 2015, pp. 18–24.

[20] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan, "Neural audio synthesis of musical notes with wavenet autoencoders," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1068–1077.