Parallel Evolutionary Optimization of Digital Sound Synthesis Parameters

Batuhan Bozkurt[1] and Kamer Ali Yüksel[2]

[1] Centre for Advanced Studies in Music,
Istanbul Technical University, Macka, 34357 Istanbul, Turkey
batuhan@batuhanbozkurt.com

[2] Computer Vision and Pattern Analysis Laboratory,
Sabanci University, Orhanli - Tuzla, 34956 Istanbul, Turkey
kamer@sabanciuniv.edu

Abstract. In this research, we propose a novel parallelizable architecture for the optimization of various sound synthesis parameters. The architecture employs genetic algorithms to match the parameters of different sound synthesizer topologies to target sounds. The fitness function is evaluated in parallel to decrease its convergence time. Based on the proposed architecture, we have implemented a framework using the SuperCollider audio synthesis and programming environment and conducted several experiments. The results of the experiments have shown that the framework can be utilized for accurate estimation of the sound synthesis parameters at promising speeds.

Keywords: computer music, parameter estimation, evolutionary computation, parallel computing.

1 Introduction

Any attempt for sound analysis is also a form of endeavor for some sort of parameter estimation [16, pg 596]. The analysis task might be undertaken for obtaining the properties of some source sound that is to be re-synthesized with different sound synthesis methods, or for observing those very qualities with the purpose of fitting them into a theoretical model. For instance, Roads [16, pg 596] points out that the Fourier Analysis can be considered as a parameter estimation method, because the results returned by such an analysis (namely magnitude and phase dissections for the analyzed signal) can be considered as parameters for a sine wave re-synthesis method that will approximate the source content veridically. However, we approach the problem of parameter estimation for effectively reducing the amount of data that is required to approximate a given sound with different synthesis methods in order to be able to control and alter various perceptual qualities of the resulting sounds from a higher level of abstraction, in an intuitive and interactive manner.

In this paper, we introduce the use of a parallelizable evolutionary architecture to optimize the parameters of sound synthesizers. The architecture is implemented as a modular evolutionary framework conceived inside the SuperCollider

(SC) programming language that specializes in audio synthesis and algorithmic composition [12,13]. The framework uses genetic algorithms (GA) to automatically optimize the set of parameters required to approximate any given target sound, using an arbitrary sound synthesizer topology created by the user. In order to test the efficiency of the framework, we have experimented with a percussion and a multiple modulator frequency modulation synthesizer for various target sounds. Finally, we have described ideas for opportunities on creative usages of evolutionary methodologies, which perform at interactive speeds in a highly connective real-time sound synthesis and algorithmic composition environment. The primary contribution of this work is the promising convergence time, which is obtained through the parallel architecture implemented using the SC environment and the simplified fitness function that preserves the perceptual quality.

2 Related Works

Evolutionary methodologies have been previously investigated by researchers to solve the problem of tone matching and parameter estimation for several different synthesizer topologies. Manzolli et al. [11] introduced the evolution of waveforms to the corresponding psychoacoustic attributes of target sounds. Horner et al. utilized evolutionary methods for parameter matching using Wavetable Synthesis [6], Frequency Modulation Synthesis (FM) [9,7,8], and Group Synthesis [2]. Garcia [5] used a genetic programming approach for automating the design of sound synthesizer algorithms represented in the form of acyclic tree graphs. Johnson [10] benefited from interactive genetic algorithms (IGA) where an user conducted interactive approach was investigated to search the parameter space and direct the parameters of the Csound FOF synthesizer.

The vast majority of the studies in this area aim to justify the suitability and efficiency of the proposed methods for the aforementioned task. For that reason, the software tools used in those works are quite specialized in demonstrating the traits of the particular proposition in which the research focuses upon (with the exception of [1] and [4] using Csound and Pure Data). Consequently, they lack connectivity with other pieces of digital music performance software; thus, obtaining and using them for compositional, live performance and other creative purposes in a practical manner is difficult.

3 Architecture

The proposed architecture consists of a host environment for sound synthesis, a modular evolutionary framework (EVWorkBench) and a parameter estimation system (EVMatch).

The host environment is implemented using [13] audio synthesis and programming language of SC (referred as *sclang*). The sound synthesis requests are handled at server-side (*scsynth*) within the SC environment through compiled unit generator graph functions (Fig. 1). This architecture allows a single sclang client to control multiple instances of local and networked scsynth servers via the Open Sound Control (OSC) protocol [17].

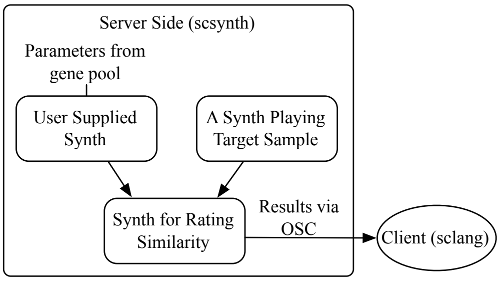
Server Side (scsynth)
Parameters from gene pool
User Supplied Synth
A Synth Playing Target Sample
Synth for Rating Similarity
Results via OSC
Client (sclang)

Fig. 1. Server-side Fitness Evaluation through OSC protocol

The initialization of the parameters and application of genetic operators are handled by the wrapped EVWorkBench class. In this way, the stages of the optimization workflow (such as initialization, selection, crossover and mutation) are separated to create a modularized framework where various evolutionary approaches can easily be experimented. Furthermore, the layered learning (having increasingly demanding fitness functions) [14] and fine tuning of the bounds of mutation operators throughout evolution are supported through interpreted and interactive language of the host environment

The EVMatch class is responsible for compilation of the synthesis definition (referred as SynthDef) provided by the user, transmission of the compiled SynthDef to the supplied servers (local and networked), the evaluation and optimization of the parameters in the gene pool, and distribution of the computational load of the evaluation across the servers registered with the instance.

In this work, the optimization is performed inside the parameter spaces of static synthesizer topologies supplied by the user for a target sound. However, it is also possible to encode synthesizer topologies as chromosomes and evolve sound synthesis algorithms as described by [5]. The speed of optimization can be increased by affecting the convergence time of the search using the direct control of the selection pressure (e.g. the tournament size) that can be tuned for different search domains and desired time constraints.

3.1 GA Optimization

To begin optimization, the user supplies a target sound file with desired attributes, a synthesizer definition, parameter names and default values of the parameters, which forms the initial population. After this initialization, the GA loop (Fig. 2) happens in generations for evolving towards better solutions of parameter sets. In each iteration, GA synthesize sounds for each set of parameter and compare the attributes of the output sound to the target sound by

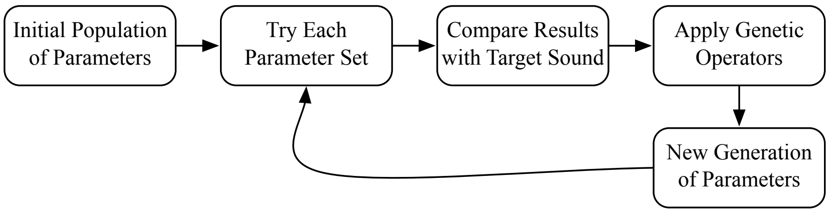
Initial Population of Parameters
Try Each Parameter Set
Compare Results with Target Sound
Apply Genetic Operators
New Generation of Parameters

Fig. 2. The GA Loop

calculating the fitness of them. Then, multiple individuals are stochastically selected from the current population based on their fitness to breed a new generation. After selection, crossover and mutation operators are applied to the gene pool. This loop continues until satisfactory fitness level, which is set by the user, has been reached for the population. In addition to a target fitness value, the user can decide the fitness of the fittest member of the last generation by listening or can limit the maximum number of generations to be iterated.

Reproduction. The tournament selection [15] is preferred concerning the relationship between the convergence rate of the GA and selection pressure, and suitability of tournament selection for noisy GA chains. Tournament selection involves running several tournaments among a few individuals chosen at random from the population. After each tournament, the one with the best fitness (the winner) is selected for crossover. Afterwards, a multi-point crossover operator where the number of split points determined proportionally to the crossover probability is applied.

4 Fitness Evaluation

The fitness evaluation provides fitness scores for each member of the gene pool that will influence the selection stage that will then steer the evolutionary process. The fitness function that we use for the parameter estimation compares the attributes of the sound output by the synthesizer running with each set of parameters inside the gene pool with attributes of the target sound.

The fitness evaluation results in a *fitness rating* that reveals the extent of similarity between two sounds. For the fitness rating, we have computed the analytical spectral distance of magnitudes between the complex spectrums is used as similarity measure of the source (synthesized) and target sound. In our implementation, we've used the analytical distance metrics proposed by [5] disregarding the phase information and focusing only on the magnitudes. The simplification of the fitness function decreases the convergence time dramatically while it preserves the perceptual quality of the synthesized sound, as it rather depends on the given synthesizer topology.

$$MSEMag = \frac{1}{Frames} \sum_{j=1}^{Frames} \sum_{i=1}^{Bins} \left[(|X_{ij}| - |T_{ij}|)^2 Wm_{ij} \right] \tag{1}$$

$$Wm_{ij} = O + (1 - O) \frac{\log|T_{ij}| - \min(\log|T_{ij}|)}{|\min(\log|T_{ij}|) - \max(\log|T_{ij}|)|} \tag{2}$$

The Eq. 1 calculates the mean squared error (MSE) between the magnitude spectrograms of synthesized and target sounds. The weight matrix WM_{ij} (Eq. 2) helps for curtailing the errors at spectral regions with more energy. The influence of the weight matrix in MSE calculation can be adjusted with O.

5 Parallelization

The fitness evaluation is parallelizable because calculations for determining the fitness scores for all members of the gene pool in a generation can be run concurrently as the fitness function only needs the chromosome of a single individual in order to work. In our work, the evaluation of the entire population is divided between servers that are registered to the instance of the parameter estimation. Thus, the workload is efficiently partitioned for independent processing across available logical processors in a single system, and computers available in a network. Thus, it handles the synchronization and the distribution of the load across registered servers automatically. The parallelization of the parameter matching enables users to work with sound at interactive speed that is suitable for digital music performances. For that reason, we have used the client-server architecture (referred as *scsynth*) of the SC that enables parallelization of the computational tasks regarding real-time sound synthesis and analysis across the CPU cores of a single machine as well as multiple networked computers.

6 Experiments

In order to determine the efficiency of the framework, we have conducted various tests in typical usage scenarios including a percussion synthesizer and multiple modulator frequency modulation topography. All of the experiments are processed using a mobile "Intel Core 2 Duo 2.4 Ghz (Penryn)" processor by utilizing both cores.

6.1 Percussion Synthesizer (PS)

To create electronic drum and percussion sounds, we have implemented a percussion synthesizer (PS) (Fig 3) that is composed of a sine oscillator with a frequency envelope acting as a simple membrane, and a filtered noise generator envelope for helping cymbal sounds or attack noises. The sum of both envelopes is fed back to the phase input of the sine oscillator (feedback PM) and it reaches

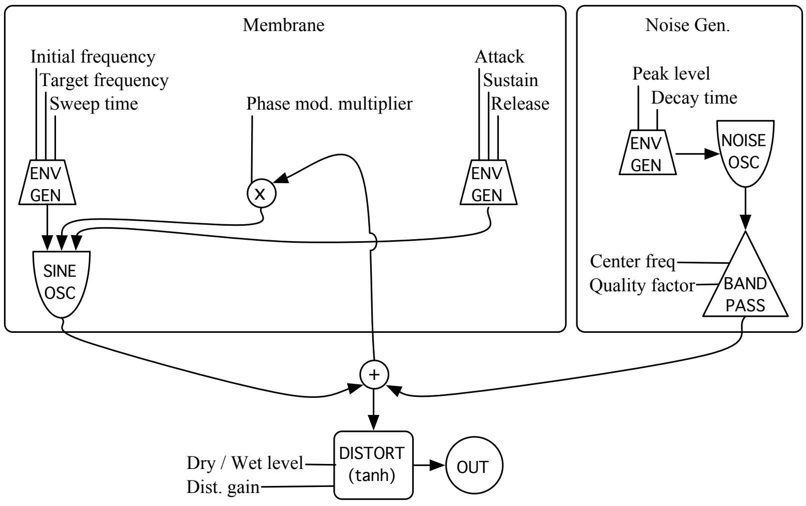

Fig. 3. Topology of the PS that creates electronic drum and percussion sounds

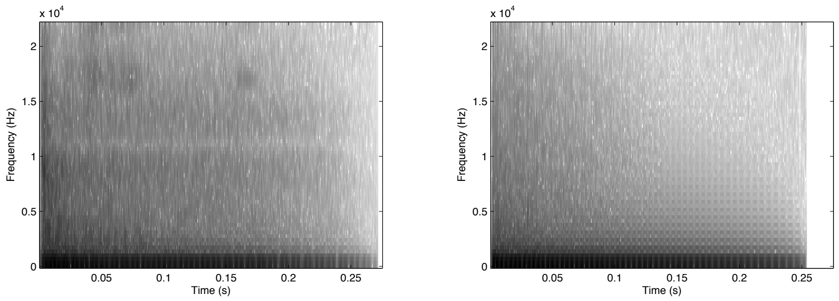

Fig. 4. Linear frequency scale spectrogram for an acoustic tom drum sample, and the output of the parameter matched by the percussion synthesizer

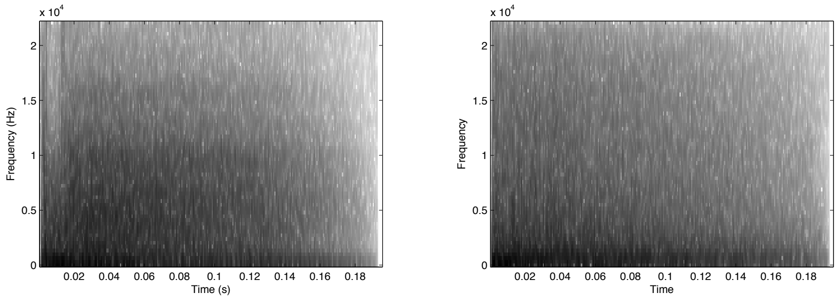

Fig. 5. Linear frequency scale spectrogram output for an acoustic snare drum sample, and the output of the parameter matched by the percussion synthesizer

the output after a distortion stage (*tanh* distortion). Almost all stages of the PS (including envelope arguments, PM amount, distortion amount, filter settings) are parameterized.

Although the PS is capable of producing a variety of electronic drum and percussion sounds, the parameter search space is not very complex for relevant sounds (such as percussive sounds that are possibly created with other drum synthesizers). However, sounds with acoustic origins have more complex spectrums; thus, PS would not be able to produce perfectly their distinctively complex spectra. Yet, we have observed that PS is capable of making cross-genre translations between sounds using our architecture. For example, PS was capable of converging to characteristic cymbal sounds that are similar to ones typically programmed for a drum synthesizer when it is fed with actual cymbal recordings for translation of that sound to an electronic version. Spectrogram views of two target (acoustic tom and snare drum samples) and synthesized sound pairs are provided in Fig. 4 and 5, where the GA converges to a fit solution in approximately 15 seconds.

6.2 Multiple Modulator Frequency Modulation (Parallel MM-FM)

A parallel MM-FM topography (Fig. 6), where 3 sine oscillators are modulating a single carrier oscillator, is implemented to test the framework in more complex search spaces. Base frequencies, modulation amounts and indexes are parametrized and no integer multiple relationships in carrier and modulator

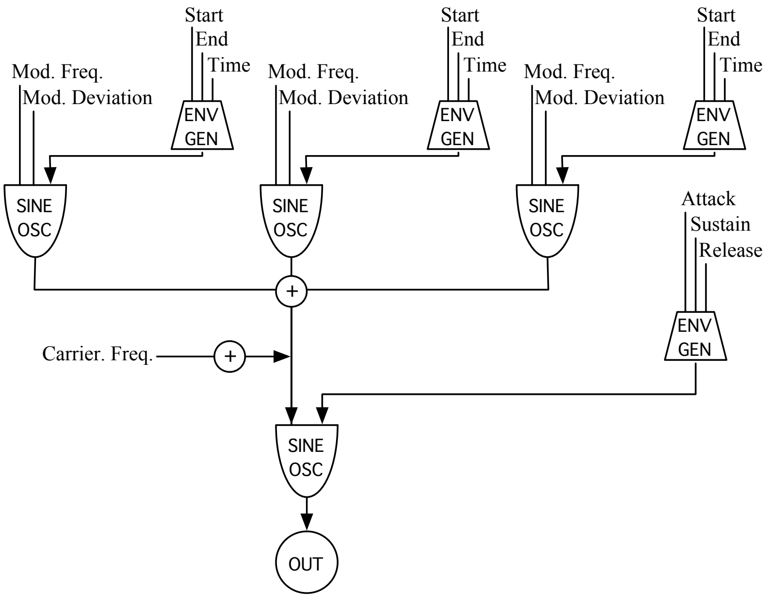

Fig. 6. Topology of the parallel MM-FM synthesizer with 3 modulators

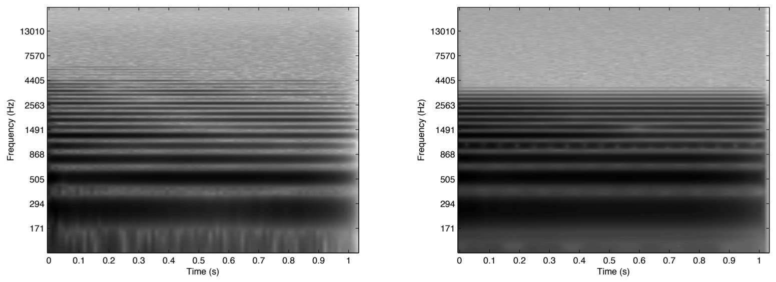

Fig. 7. Logarithmic frequency scale spectrogram output for a recorded piano sample and the output of the parameter matched synthesis by the MM-FM synthesizer

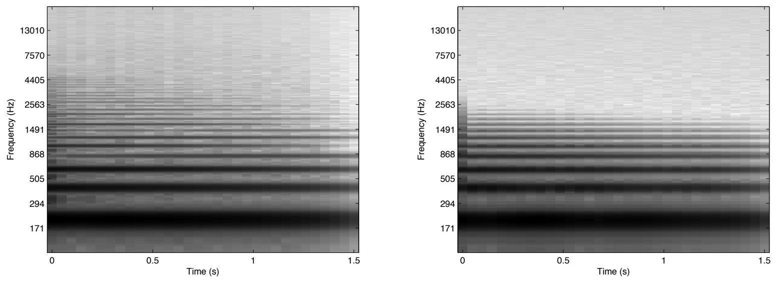

Fig. 8. Logarithmic frequency scale spectrogram output for a recorded rhodes keyboard sample and the output of the parameter matched synthesis by the MM-FM synthesizer

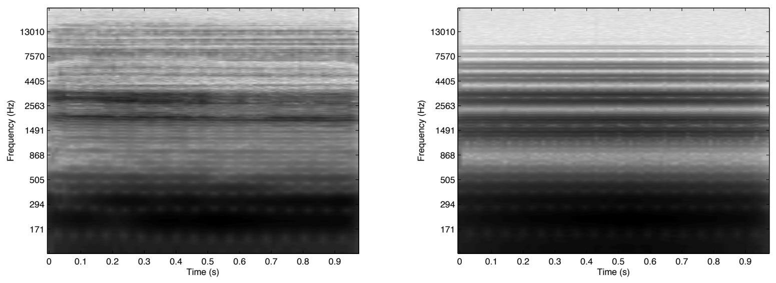

Fig. 9. Logarithmic frequency scale spectrogram output for a recorded human voice producing the "ee" vowel and the output of the parameter matched synthesis by the MM-FM synthesizer

frequencies is implied in the design. The MM-FM synthesizer is capable of producing variety of complex harmonic and inharmonic spectra. Various harmonic target sounds have been experimented using variety of GA settings; yet, the average convergence time is considerably higher (around 3 minutes) than PS because the search space is significantly more complex. When C:M ratio is locked to integer multiple relationships (without frequency envelopes), convergence time decreases substantially; however, the produced sounds become much less interesting. Spectrogram views for various target (including piano, rhodes keyboard and human voice samples) and synthesized sound pairs are provided in Figures 7, 8 and 9 respectively.

7 Discussion

The implementation was intended to be apposite for general-purpose search optimization tasks; thus, the parameter estimation system does not rely on a particular synthesis technique or synthesizer topology to be functional. Hence, the complexity of the search space is influenced directly by parameter ranges defined for the synthesizer in relation with the target sound. Paying special attention to providing possible parameter ranges is not strictly necessary, as the GA search will eliminate unintelligible parameters for a given topology. However, directing the algorithm to a set of possibly related parameter ranges would greatly decrease the complexity of the search; thus, better results might be obtained in a fixed time window. Using such setup, it might be possible to compute a parameter setting, which yields good results with virtually any sound produced by an instrument, given a few sound samples of it.

However in practical terms, there is no guarantee for an arbitrary provided method to approximate a target sound satisfactorily [16, pg 596] and usually the GA alone may not able to eliminate sets of parameters that do not represent feasible solutions. Fortunately, experimenting small or large scale synthesizer topologies is relatively easy thanks to unit generators provided within the SC distribution. Thus, synthesizers are available for sound generation immediately after defining the unit generator graph function. The process is so robust that live programming and control of synthesizers are handled real-time in live coding performances by various computer musicians [3]. The simplicity in dynamic, interactive and almost improvisatory synthesizer creation yields to some interesting opportunities like easy experimentation with cross-genre transformation of sounds and errant approaches to parameter estimation between seemingly unfit synthesizers and targets for creative purposes.

8 Conclusion

In this work, we have proposed a general-purpose evolutionary framework, which is integrated into the SuperCollider (SC) software platform, to perform flexible parameter estimation for digital sound synthesizers. The system is able to provide solutions for parameter estimation tasks at interactive speeds typically necessitated for live performance and composition workflows. The modular and flexible structure of the framework may open wide-range of opportunities for musicians and researchers in the field.

References

1. Boulanger, R.C.: The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming. MIT Press, Cambridge (2000)
2. Cheung, N.M., Horner, A.B.: Group Synthesis with Genetic Algorithms. Journal of the Audio Engineering Society 44(3) (1996)
3. Collins, N., McLean, A., Rohrhuber, J., Ward, A.: Live Coding in Laptop Performance. Organised Sound 8(3), 321–330 (2003)
4. Fornari, J.: An ESSynth Implementation in PD. In: SIGGRAPH 2006: ACM SIGGRAPH 2006 Research Posters, p. 106. ACM, New York (2006)
5. Garcia, R.: Growing Sound Synthesizers Using Evolutionary Methods. In: European Conference in Artificial Life ECAL 2001. Artificial Life Models for Musical Applications, University of Economics, Prague (2001)
6. Horner, A.: Wavetable Matching Synthesis of Dynamic Instruments with Genetic Algorithms. Journal of Audio Engineering Society 43(11), 916–931 (1995)
7. Horner, A.: Nested Modulator and Feedback FM Matching of Instrument Tones. IEEE Transactions on Speech and Audio Processing 6(4) (1998)
8. Horner, A.: Double-Modulator FM Matching of Instrument Tones. Computer Music Journal 20(2), 57–71 (1996)
9. Horner, A., Beauchamp, J., Haken, L.: Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis. Computer Music Journal 17(4), 17–29 (1993)
10. Johnson, C.: Exploring Sound-Space with Interactive Genetic Algorithms. Leonardo 36(1), 51–54 (2003)
11. Manzolli, J., Maia Jr., A., Fornari, J., Damiani, F.: The Evolutionary Sound Synthesis Method. In: MULTIMEDIA 2001: Proceedings of The Ninth ACM International Conference on Multimedia, pp. 585–587. ACM, New York (2001)
12. McCartney, J.: SuperCollider, a New Real Time Synthesis Language. In: Proceedings of the International Computer Music Conference, pp. 257–258 (1996)
13. McCartney, J.: Rethinking the Computer Music Language: SuperCollider. Computer Music Journal 26(4), 61–68 (2002)
14. McDermott, J., O'Neill, M., Griffith, J.L.: Target-driven Genetic Algorithms for Synthesizer Control. In: 9th Int. Conference on Digital Audio Effects, DAFx 2006 (2006)
15. Miller, B.L., Goldberg, D.E.: Genetic Algorithms, Tournament Selection, and the Effects of Noise. Complex Systems 9(3) (1995)
16. Roads, C., Strawn, J.: The Computer Music Tutorial, 4th edn. MIT Press, Cambridge (1999)
17. Wright, M.: Open Sound Control: An Enabling Technology for Musical Networking. Organised Sound 10(3), 193–200 (2005)