SUPPORT FOR LEARNING SYNTHESISER PROGRAMMING

Mateusz Dykiert
University College London
m.dykiert@cs.ucl.ac.uk

Nicolas Gold
University College London
n.gold@ucl.ac.uk

ABSTRACT

When learning an instrument, students often like to emulate the sound and style of their favourite performers. The learning process takes many years of study and practice. In the case of synthesisers the vast parameter space involved can be daunting and unintuitive to the novice making it hard to define their desired sound and difficult to understand how it was achieved. Previous research has produced methods for automatically determining an appropriate parameter set to produce a desired sound but this can still require many parameters and does not explain or demonstrate the effect of particular parameters on the resulting sound. As a first step to solving this problem, this paper presents a new approach to searching the synthesiser parameter space to find a sound, reformulating it as a multi-objective optimisation problem (MOOP) where two competing objectives (closeness of perceived sonic match and number of parameters) are considered. As a proof-of-concept a pareto-optimal search algorithm (NSGA-II) is applied to CSound patches of varying complexity to generate a pareto front of non-dominating (i.e. "equally good") solutions. The results offer insight into the extent to which the size and nature of parameter sets can be reduced whilst still retaining an acceptable degree of perceived sonic match between target and candidate sound.

1. INTRODUCTION

When learning an instrument, students often like to emulate the sound and style of their favourite performers. The learning process takes many years of study and practice and yet for software synthesisers, the user must deal with complexities involved in specification of the timbre [1]. The vast parameter space makes it challenging for the inexperienced user to achieve a desired sound as a small change in one parameter can alter the result quite significantly. Several authors describe approaches to automatically determine a set of synthesiser parameters that can produce a desired target sound (see for example Horner [2], Johnson [3], Lai [4], and McDermott [5]). However, achieving a target sound does not necessarily help a user to learn the effect of particular parameters. What is required is both a way of achieving desired sounds, and also guiding a user in meaningful experimentation around that sound to aid learning. This will require the development of underlying technology for automatic parameter derivation and the development of appropriate pedagogy and supporting tools.

As a first step towards this goal, this paper presents a reformulation of the synthesiser sound-matching problem as a Multi-Objective Optimisation Problem (MOOP) where two objectives are considered: closeness of sonic match, and number of parameters used to achieve the match. Maintaining an acceptable sonic match to the target sound while reducing the number of parameters required to attain it should aid an inexperienced user since the apparent complexity of the synthesiser is reduced. Determining the appropriate balance between the size of the parameter set and the closeness of match is a difficult problem as the trade-off may not be easy to judge (for example, it might be that there are parameter sets that are considerably smaller than the original but that do not substantially affect the sound, or in other cases it may be that a single parameter can have a major impact on the timbral quality). To gain insight into these trade-offs and inspired by an analogous problem in software engineering (see [6]), we have applied a multi-objective pareto-optimal search algorithm (NSGA-II). Such algorithms generate a pareto front of non-dominating (i.e. equally-good) solutions. Plotting these on a graph then allows the trade-offs between the size of parameter set and sonic match to be more easily seen.

The paper makes the following contributions:

1. A reformulation of the search for synthesiser parameter sets as a multi-objective search problem.

2. A solution representation for encoding parameters and fitness for use by the NSGA-II algorithm.

3. The results of an empirical study of the NSGA-II algorithm applied to a number of CSound synthesisers.

The remainder of the paper is organised as follows. Section 2 presents the reformulation of parameter search as a MOOP and introduces relevant theory. Section 3 describes the solution representations. Section 4 describes the experimental configuration that led to the results discussed in Section 5. Section 6 presents related work and Section 7 concludes.

2. RELATED WORK

There are several applications of genetic algorithms to the generation of synthesiser parameters to produce a sounds similar to a given target.

Yee-King and Roth [7] present a software synthesiser programmer. Their Java-based system is capable of automatically programming any VSTi compatible software synthesiser. A genetic algorithm provides the core of this system with parameters encoded as real numbers from 0 to 1 into a chromosome. The system uses proportionate roulette wheel selection and uniform random crossover as genetic operators. Mutation is achieved by adding a gaussian random variable. The fitness function is based on MFCCs compared between target and candidate solution to produce Square Root Mean Error. The system has been evaluated using expert users and produces close perceptually matching sounds. We adopt the same fitness approach for the calculation of sonic distance but our approach differs in the consideration of the second fitness objective (number of parameters), and the GA operators.

McDermott et al. [5] also use a genetic algorithm to deduce the set of synthesiser parameters. A comprehensive study of different fitness functions is presented, including pointwise metric, perceptual metric (made up from centroid, harmonicity and attack time), discrete Fourier transform metric and composite metric derived from simpler measures. They observe that all of these fitness functions perform well on simple target, but their performance is highly diminished when working on targets containing many partials. Their conclusion is that the perceptual measure is the most suitable for such computation. Our approach follows this finding in using a perceptually driven metric (although we use MFCC rather than the composite metric suggested) but differs in that our aims is to reduce the size of the parameter set.

Lai et al. [4] present another solution based on genetic algorithms for FM synthesis. The system is evaluated using different fitness functions. The sonic match is determined using spectral centroid and spectral norm derived from short time Fourier transform. Spectral centroid is found to perform better than spectral norm used alone, but much better results are obtained if these two functions are combined together. The system is evaluated against a piano tone generated by a Yamaha MA3 FM synthesiser with known pitch. Closely matched sounds are reported to be generated. Our work shares a similar overall approach but again, we differ in that we are undertaking a multi-objective approach and are currently focusing on additive synthesis (although FM would be an interesting avenue of future work for our approach).

3. PROBLEM DEFINITION

This section defines the sonic matching problem more formally as a MOOP. Multi-objective optimisation problems have two or more equally important objectives to be independently maximised (or minimised) simultaneously [8].

A synthesiser $\mathcal{S}$, uses a set $\mathcal{P}$ of parameters to generate a sound. Each parameter n_p is an integer in the range 0-127 (corresponding to the possible MIDI CC values sent, for example, from a control surface). Although this restricts the possible values available for each parameter, the combination is still a large search space. If the parameters were not restricted, many more iterations would be required before the algorithm converged on a set of good solutions.

An audio fragment $\mathcal{A}$ is a 0.5 second fragment of audio of constant amplitude and pitch. The distance d between two sounds $\mathcal{A}_1$ and $\mathcal{A}_2$ is defined thus:

$$d = \sqrt{\frac{\sum_{i=0}^{N}(x_{1,i} - x_{2,i})^2}{n}}, \quad (1)$$

where n is the number of parameters, x_1 is the $\mathcal{A}_1$'s MFCC and x_2 the $\mathcal{A}_2$'s MFCC value.

The problem of sound matching with fewer parameters can therefore be seen as one of minimising d while simultaneously minimising the arity of $\mathcal{P}$.

3.1 Genetic algorithms

Genetic Algorithms (GAs) were first introduced by Holland in 1975 [9]. They are inspired by biological processes of evolution - selection, mutation, crossover, for optimising and solving complex problems. Genetic algorithms generate a random initial population which is then evolved through reproduction, forming new populations until a stopping condition is reached e.g. a specific number of function evaluations or the solution of required fitness is found.

3.1.1 Representation

The candidate solution is encoded into a chromosome. The way that the values are encoded in a chromosome is implementation dependent. Usually values are encoded as a binary string.

3.1.2 Selection

Selection is used for determining parents used in crossover. There exist many selection operators, such as roulette wheel and tournament [9].

3.1.3 Crossover

Crossover exchanges selected genes from parents obtained in selection. New offspring are created in this step [9].

3.1.4 Mutation

In order to maintain diversity of solutions within generations, the mutation operator randomly changes a gene of newly created offspring during crossover [9].

4. SOLUTION

This section describes the algorithms used and provides more detailed insight into our approach.

4.1 Parameter Encoding

Synthesiser parameters can have different types and value ranges. This introduces complexities for genetic algorithms due to the requirement of different type encodings within the chromosome. The complexity is carried forward to mutation and crossover operators, which need to make sure that the particular operation does not create invalid (out of range) solutions. Following certain standards can reduce

this complexity. For example, Yee-King and Roth [7] use a VST compatible backend, hence parameters are encoded as real numbers from 0 to 1. We follow a similar idea, adopting the MIDI specification where most continuous controllers conform to a range of integers between 0 and 127. This uniform way of storing values ensures that the parameter is always valid and within the range specified. A secondary list of parameters is maintained for lookup of maximum and minimum values for the further value projection required for generating candidate solutions.

In the case of the CSound patches used, we need to project the value into a real parameter range. In order to achieve this we use the following equation:

$$t = \frac{(a-b)}{128}.v + b, \tag{2}$$

where t is the actual parameter value, a is the actual maximum parameter value, b is the actual minimum parameter value and v is the value stored in a chromosome.

4.2 Fitness Assignment

With the transformation of the problem into multi-objective optimisation problem, two fitness functions must be defined. This section describes the two fitness functions used in the test system.

4.2.1 Sonic Match

The first objective needs to be evaluated to classify how similar the candidate and target sounds are. In order to achieve this, we use Mel-Frequency Cepstral Coefficients (MFCC). The concept was originally introduced as measures for speech recognition [10] but its application in music has been growing ever since MFCCs provide perceptually meaningful means for the classification of audio signals.

MFCCs are computed in four steps [11]:

1. Applying the discrete Fourier transform on a windowed sound segment.
2. Mapping powers from resulting spectrum onto Mel scale.
3. Convolving warped power spectrum with triangular band-pass filter and taking natural logarithm of the result.
4. Final computation of MFCC using Equation 3.

Davis and Mermelstein in [12] express the computation of MFCCs as:

$$MFCC_i = \sum_{k=0}^{20} X_k cos\{i(k-\frac{1}{2})\frac{\pi}{20}\} \quad , k = 1, 2, ..., M, \tag{3}$$

where M is the number of cepstrum coefficients and X_k is the low-energy output of kth filter.

In a similar approach to the work of Yee-King and Roth [7], the fitness is expressed as the RMS Error of the MFCCs.

4.2.2 Number of Parameters

The second objective function must determine when a particular parameter has no effect on the sound generated during computation (in other words, for all practical purposes, that parameter is redundant). Each parameter has a value which causes no effect on the sound. These values are specific to the individual parameters, for example, a high-pass filter will not have any effect if the cut-off frequency is below the level at which humans can perceive sound. To calculate the number of parameters which actually have any effect, we sum all parameters for which the no-effect condition is false.

4.3 NSGA-II

Algorithm 1 shows the NSGA-II algorithm defined by Deb et al. [13].

Initially, a random parent population P_0 is created. Solutions are evaluated for fitness. Offspring population $\mathcal{Q}_\prime$ is created by selection using binary tournament, crossover and mutation.

Algorithm 1 NSGA-II - main loop

```
while stopping condition not met do
  R_t = P_t ∪ Q_⊔
  F = fast-non-dominated-sort(R_t)
  P_{t+1} = ∅ and i = 1
  while |P_{t+1}| + |F_⟩| ≤ N do
    crowding-distance-assignment(F_⟩)
    P_{i+1} = P_{t+1} ∪ F_⟩
    i = i+1
  end while
  Sort(F_⟩, ≺_n)
  P_{t+1} = P_{t+1} ∪ F_⟩[1 : (N − |P_{t+1}|)]
  Q_{t+1} = make-new-pop(P_{t+1})
  t = t+1
end while
```

NSGA-II introduces elitism to maintain the best solutions found so far. Each individual in the population (i.e. a set of parameter values) is assessed for the number of effective parameters and sonic match closeness. A fast non-dominated sorting algorithm is used to sort the population into different fronts. Each front has a different non-dominated rank, hence NSGA-II algorithm assigns a special fitness value to each solution according to the front on which it is located. Crowding distance is used as a second internal fitness value assigned according to the magnitude of the distance. Crowding distance is said to be an density estimate of the front.

4.4 Random Search

To provide a comparison with NSGA-II, we have also run the experiments using the Random Search (RS) algorithm in jMetal [14]. This technique was used to check the validity of the formulated problem, as NSGA-II should have no problem outperforming random search.

Name	# Params	Filter 1	Filter 2	Filter 3
S1	20	-	-	-
S2	21	Low-Pass	-	-
S3	21	High-Pass	-	-
S4	22	Band-Pass	-	-
S5	22	Low-Pass	High-Pass	-
S6	24	Low-Pass	High-Pass	Band-Pass
S7	24	Band-Pass	Low-Pass	High-Pass

Table 1. This table describes filters used within CSound patches and the number of parameters which the patch accepts.

5. EXPERIMENTAL SETUP

Each experiment used a different target sound which was randomly generated using the same CSound patch. Each experiment was executed 8 times for each CSound patch for both NSGA-II and Random Search. The length of the target and all generated sounds was limited to 0.5s. The amplitude and fundamental frequency are constant.

In our system we are using MFCC implementation in Java from Comirva project [15] and the NSGA-II implementation of jMetal [14].

The CSound patch used in the experiments is made up from two oscillators which use composite waveforms generated by functions using GEN10 routines. Each function specifies 10 relative strengths of each fixed harmonic partial. The composite waveforms created are added together to produce a sound. This represents a CSound patch called S1. The Table 1 outlines the different orchestra files used by the system.

5.1 EA Parameters and Operators

Each experiment has been run for 5000 fitness function evaluations. The initial population size was set to 100, and similarly, the archive population was also set to 100. All experiments used the same operators: single point crossover, bit flip mutation and tournament selection. All experiments were run with crossover probability $P_c = 0.7$ and mutation probability to $P_m = 1/n$ (where n is the number of parameters). Synthesiser parameters were encoded as integers, in range from 0 to 127, to provide uniform and MIDI compatible representation in a chromosome.

6. RESULTS

In this study we investigate the feasibility of our concept to reduce the number of parameters necessary to achieve a desired sound. In order to achieve this, we use the seven different CSound patches outlined in Table 1. The patches have a similar base structure, but they differ in complexity. Patches S6 and S7 contain the same components, but different arrangements and signal routing.

The results are shown in Figure 1. Each subfigure shows the NSGA-II obtained non-dominated front with all solutions generated by Random Search algorithm (the point cloud in each diagram). The lines joining points on the pareto front are not themselves meaningful but are simply included to show the front more clearly. Subfigures a-g represent typical runs of the experiment (5000 function evaluations) on seven patches described. Figure 2 presents results for the S1 patch over 20000 function evaluations. Clearly, the increased number of function evaluations increased the closeness of the match and minimised the number of parameters more than an experiment with 5000 function evaluations. This situation is not always the case with other patches - sometimes the sonic match cannot be better, and the number of parameters required is not minimised further.

The results obtained show that the NSGA-II algorithm usually obtains a solution with a closer sonic match to the target than Random Search. NSGA-II is also better in minimising the number of parameters having an effect on the generated sound, finding such solutions more quickly and with more success. The lower extreme of the parameter number of parameter space is explored much further than with Random Search, converging on more suitable solutions. On average NSGA-II finds a solution requiring around two parameters less than equivalent solution obtained by RS. Subfigure Figure 1(a) clearly shows a good example of this; comparing similar sonic match values, the best solution obtained by NSGA-II algorithm has 17 parameters whereas the Random Search solution requires 20.

The repeated runs of the same experiment are fairly consistent and follow a similar trend.

The NSGA-II solution will scale better, judging by experiments involving S6 and S7 as it handles parameter optimisation better in more complicated patches. Both S6 and S7 have 24 parameters and the number of parameters is reduced the most in comparison to other simpler experiments involving 18 parameters.

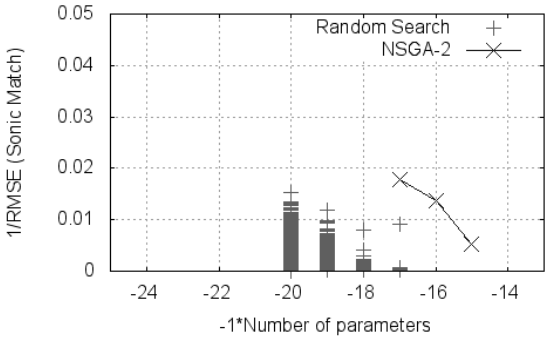
In rare cases, the random search has produced a singular very, very good solution with a large number of parameters.

Extended runs of the experiments, including 10k and 20k function evaluations, have shown that the NSGA-II algorithm outperforms Random Search. With increased number of iterations, RS has improved slightly, and some solutions with reduced number of parameters are obtained.

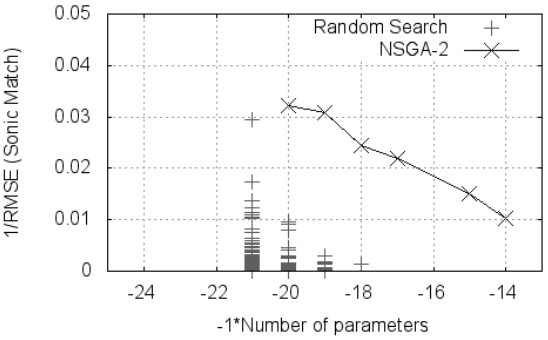
The performance of the test system is fairly good and consistent. The time it takes to perform an experiment with 5000 iterations is around 6 minutes on a machine with Intel Core i3 2.4GHz processor with 4GB RAM. The running time increases linearly to 13 minutes for 10k and 27 minutes for 20k function evaluations.

7. CONCLUSIONS AND FUTURE WORK

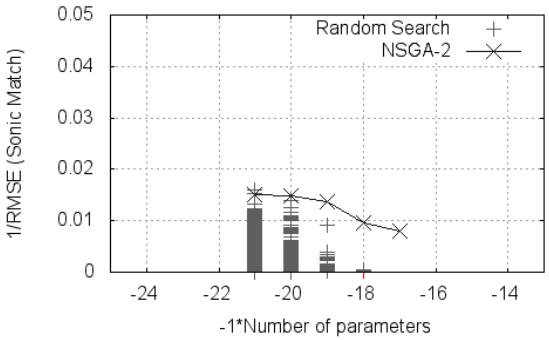
This paper has presented a reformulation of the search for synthesiser parameters as a multi-objective optimisation problem to reduce the complexity of synthesiser programming. Two objectives were considered: the closeness of sonic match (in common with other previous methods in this area), and the number of effective parameters in the



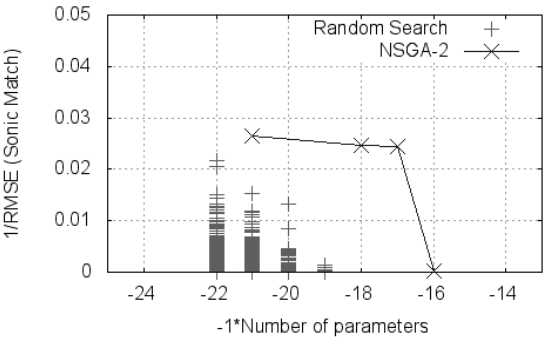
(a) S1 patch, max 20 parameters



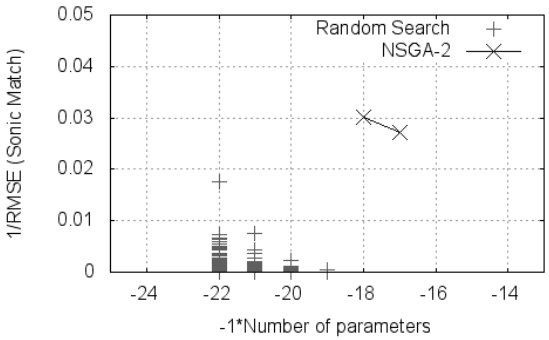
(b) S2 patch, max 21 parameters



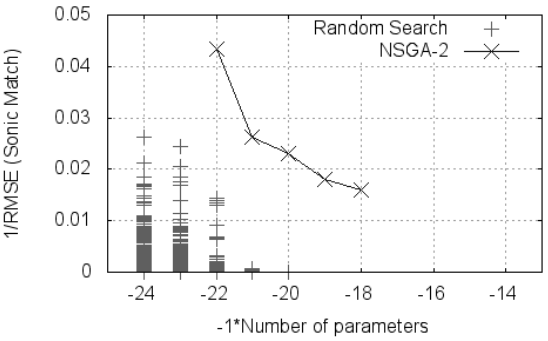
(c) S3 patch, max 21 parameters



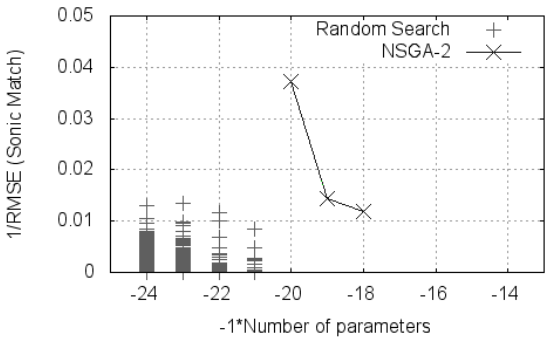
(d) S4 patch, max 22 parameters



(e) S5 patch, max 22 parameters



(f) S6 patch, max 24 parameters



(g) S7 patch, max 24 parameters

Figure 1. Graphs showing pareto front obtained by NSGA-II vs solutions generated by Random Search algorithm

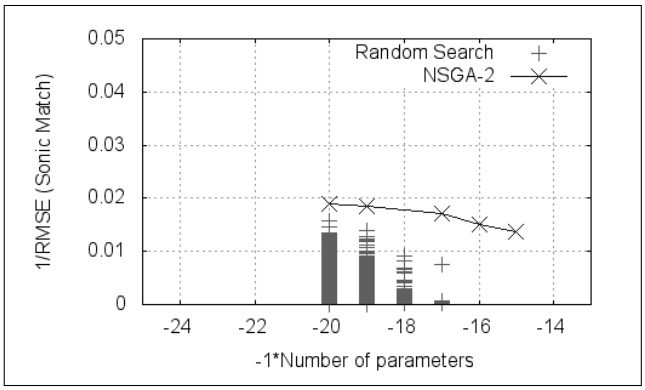


Figure 2. S1 experiment, 20k function evaluations.

generating set. A state of the art pareto-optimal search algorithm (NSGA-II) was applied to generate a pareto front of non-dominant solutions. The results were also compared to Random Search.

The results indicate that it is possible to reduce the size of the parameter set whilst still maintaining a good match to the target sound. Using a pareto-optimal search algorithm to generate pareto fronts allows insight into the tradeoff between the size of parameter set and the closeness of match, showing where further reductions in the size of the set make large differences to the match distance. The use of a single synthesiser (CSound) means that it is possible that results may not generalise to others. This is counterbalanced by the fact that the synthesis method used here is standard additive synthesis and the parameters are expressed in a MIDI compatible form, amenable to use with other synthesisers.

There are a number of directions for future work. Further investigation of the content of patches on the pareto front is expected to offer more comprehensive insight into the parameters that are most easily eliminated. Other standard search algorithms will be tried to improve the efficiency of the pareto front generation. It is possible that a hybrid co-evolutionary approach of genetic programming and genetic algorithms will offer the possibility of further simplification by modifying the architecture as well as the parameter set. More complex synthesiser architectures will also be investigated (for example, the DX7 implementation in CSound) and subsequently non-CSound synthesisers programmed via MIDI.

Acknowledgments

We thank the members of CREST (in particular Shin Yoo and Mark Harman) for useful discussions and are grateful to the Computer Science Department at University College London for supporting this work.

8. REFERENCES

[1] R. Boulanger and J. Ffitch, "Teaching software synthesis through Csound's new modelling opcodes," in *Proceedings of the International Computer Music Conference*, 1998.

[2] A. Horner, J. Beauchamp, and L. Haken, "Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis," *Computer Music Journal*, vol. 17, no. 4, pp. 17–29, 1993.

[3] C. Johnson, "Exploring the sound-space of synthesis algorithms using interactive genetic algorithms," in *Proceedings of the AISB Workshop on Artificial Intelligence and Musical Creativity, Edinburgh*, 1999.

[4] Y. Lai, S. Jeng, D. Liu, and Y. Liu, "Automated optimization of parameters for FM sound synthesis with genetic algorithms," in *International Workshop on Computer Music and Audio Technology*, 2006.

[5] J. McDermott, N. Griffith, and M. O'Neill, "Toward user-directed evolution of sound synthesis parameters," *Applications on Evolutionary Computing*, pp. 517–526, 2005.

[6] Y. Zhang, M. Harman, and S. Mansouri, "The multi-objective next release problem," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 1129–1137.

[7] M. Yee-King and M. Roth, "Synthbot: An unsupervised software synthesizer programmer," in *Proceedings of the International Computer Music Conference*, 2008.

[8] K. Deb, *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001.

[9] J. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* The University of Michigan Press, Ann Arbor, 1975.

[10] P. Mermelstein, "Distance measures for speech recognition, psychological and instrumental," *Pattern Recognition and Artificial Intelligence*, vol. 116, 1976.

[11] F. Zheng, G. Zhang, and Z. Song, "Comparison of different implementations of MFCC," *Journal of Computer Science and Technology*, vol. 16, no. 6, pp. 582–589, 2001.

[12] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[14] J. Durillo, A. Nebro, and E. Alba, "The jMetal framework for multi-objective optimization: design and architecture," in *IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 4138–4325.

[15] M. Schedl, “The CoMIRVA Toolkit for Visualizing Music-Related Data,” Department of Computational Perception, Johannes Kepler University Linz, Tech. Rep., June 2006.