Genetic Music

Kyle Harrison (4274247)

COSC 4P98
Instructor: Dr. B. Ross
Brock University
Winter 2012

Abstract

This document is meant to support the library entitled *GeneticMusicLib* as well as the GUI provided alongside it. *GeneticMusicLib* is a library which was created that generates sounds using a genetic algorithm (GA). The fitness of the created sounds is interactive, and the user is required to provide this value. It then proceeds to combine individuals to evolve the original sound in hopes of producing something more appealing to the user. This report will demonstrate the theoretical ideas behind the library, the design details of the genetic algorithm used, and an overview of how to use the application.

Contents

1 Introduction

The main component of this application is the library entitled *GeneticMusicLib*. This is developed as a class library for the .NET framework, more specifically .NET version 4. This library makes use of a freely available (for non-commercial use) audio library entitled BASS[1]. Since the original library was not designed for .NET, a third-party application programming interface (API) entitled BASS.NET, also available from the un4seen website[1], was used to allow programming in C# 4.0.

This report will give a high level description of the capabilities of the application in Section 2. Section 3 will give an overview of GAs and other theoretical aspects involved. Section 4 will give more technical details on the implementation of the GA as well as other library functions, including those provided by BASS[1]. Finally, Section 5 will serve as a user manual describing how to use the application.

2 Description of the Application

GeneticMusicLib is a library which has been developed to create genetic music; music created with the assistance of a genetic algorithm. The library provides an easy and efficient way to create short-length samples of audio which are evolved using a slightly modified GA, which will be described in greater detail in Section 4. The samples are represented using a list of frequencies for the tones, each of which is assigned a duration corresponding to a musical note (quarter, eighth, etc.). Having notes of varying length adds an extra level of dynamic to the samples.

The GUI portion of the application serves as a wrapper for the library, allowing far easier access to the functionality. When the loads the application, they are presented with a number of parameters, described in greater detail in Sections 4 and 5. The GUI allows the user to create an initial population of chromosomes, then continuously generate new populations using standard GA parameters. There is functionality to playback the audio created by the GA, on the default system audio device, or saving the generated samples, in WAV format, by making use of the BASS[1] libraries.

3 Background

The idea for this project stems from an interest in artificial intelligence and evolutionary computation. The realization that musical samples can be easily represented in GA friendly way made the decision for this project quite easy. A GA is a relatively straightforward implementation of an evolutionary algorithm which attempts to mimic natural selection. They are very effective as balancing exploration and exploitation through the use of crossover and mutation, respectively. This allows for a very large space to be effectively and efficiently searched. The high-level idea of a GA is that is carries a population of candidate solutions, called chromosomes, which are then bred and mutated in order to increase the overall fitness of the population. Pseudocode for a basic GA can be seen in Algorithm 1.

The population of chromosomes are normally initialized randomly, to give a diverse representation of the search space. Sometimes these may be seeded in areas known to be of good fitness. Since the goal here is to generate random samples, no seeding is

performed. These solutions are then ranked based on fitness, normally implemented through some form of heuristic function. Since audio is purely subjective, an interactive version is used for this application. This means that the fitness is not calculated but is rather supplied by the user. The user is providing high fitness values for sounds which they find appealing. This allows the structure of the better audio to be more closely copied into the new samples being generated.

Once the fitness has been assigned to each member of the population, the generation of a new population can begin. During the generating of a new population, there are 3 techniques which are employed: crossover, mutation, and elitism. Each of these are performed with some probability.

Crossover is the "breeding" stage, where parents are selected, based on fitness, and are recombined to form new chromosomes. Since selection pressure is put on better fit individuals, commonly through tournament or rank-based selection, the areas of high promise in the search space are explored in more depth than other regions. The breeding allows the genetic information from the parents to be kept in new generations. This is an attempt at preserving the good genetic material, while still keeping a diverse set of individuals. The crossover step is an example of an exploration based search technique, as it makes large jumps in the search space. Mutation is performed on the offspring of crossover, with some probability.

Mutation is meant to make small changes to the chromosomes, again in an attempt to increase diversity of the population. Mutation introduces patterns that were not present in the parents, but does so in such small increments that provides an exploitation based search. The mutation is meant as an attempt to perform a more fine-tuned search of the high-promise ranges.

Finally, elitism refers to copying a proportion of the best fit individuals to the new generation. This ensures that the best solutions are not lost during recombination. For the purposes of this application, the elitism is simply a single solution that is carried into the new generation. This can be enabled or disabled as it is not necessary, however it can improve the convergence onto a good solution.

Algorithm 1 Pseudocode for a simple genetic algorithm

```
function simpleGA
    Initialize chromosomes
    Evaluate fitness of each chromosome
    while termination criteria not met do
        Select best chromosomes
        Perform crossover on chosen chromosomes
        Perform mutation on created offspring
        Replace population with new generation
    end while
    return Best fit chromosome
end function
```

4 Implementation Details

The GA used in the application is slightly modified from a standard GA in one main way. That is, the fitness of a music sample is entirely subjective and thus no fitness function is used to rank the solutions. The solutions must therefore be rated manually, by the user. Each sample is played back rated by the user, and is given an integer value between 0 and 100; 0 being the lowest valued, least desirable sound and 100 being the highest valued, most desirable sound of the generation. These values hold no meaning other than the subjective ranking of the user. The fitness values are relative to each other and are only used for elitism and parent selection for crossover; having 2 samples ranked 51 and 50 as opposed to 61 and 60 bears no immediate significance to the application.

Another important difference is that the GA used does not run for a specified number of iterations. It simply presents a number of audio samples, allows the user to listen and asses the fitness, then create a new generation using the assigned fitness values. Thus, there is no stopping criteria for the GA, and it runs as long as the user wishes to continue creating new generations.

4.1 Chromosome Representation

The chromosome is represented using 2 arrays of equal length. The first array is an array of integers which correspond to the frequency of the tones in the sample. The second array is one of an enumerated type representing the length of the tones in the first array. The possible values of these are based on the standard musical notes:

1. Half note
2. Quarter note
3. Eighth note
4. Sixteenth note

These 4 note lengths were chosen as they provide a good range of durations without making individual tones too long or too short. However, these durations are also dependent on the tempo, in beats per minute (BPM), that is supplied by the user. The higher the BPM, the faster the tempo of the music; a half note at 60BPM is much longer than a half note at 120BPM.

Since we now have a way of representing both pitch and duration, we have designed a representation of a single-melody piece of music. Using only these two arrays makes it well suited for a GA, having only to worry about performing the operators on a single extra array as opposed to the standard GA. For the simplicity, and ease of adaptation, it can be stated that this is an effective way of representing the samples.

The chromosomes also have methods to produce a single integer array which completely represents the audio data. This is done by creating a single period of a sine wave at the specified frequency, then sampling this wave for the specified duration to give a constant tone. This method is designed to work specifically with the BASS libraries stream playing functionality.

4.2 Crossover Operators

The crossover operators implemented are ones that were learned in COSC 3P71 (fall 2010 with Dr. Ombuki) and are all 2-parent crossovers.

4.2.1 One Point Crossover

One point crossover is the simplest crossover that can be formulated. It picks an arbitrary point in the chromosome and copies alleles before the point to the corresponding child, then alleles after the point to the other child. This is a very simplistic crossover technique which makes no attempt to vary the structure of the parents, but just copies it directly. An example of one point crossover can be seen in Figure 1. More information on one point crossover can be found on Wikipedia[2].

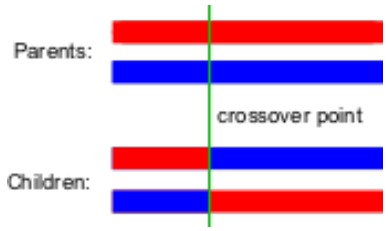

Figure 1: An example of one point crossover, obtained from `http://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)`

4.2.2 Two Point Crossover

Two point crossover is a slightly more complex crossover than the one point, but still uses the same idea. It selects two arbitrary points in the chromosomes which will be swapped. Alleles outside the two point range are copied directly to the corresponding children and points within this range are swapped with the other parents. This provides a slightly better exploration of the search space as it is now altering the structure of the chromosomes a bit as well. This introduces new patterns into the population. An example of two point crossover can be seen in Figure 2. More information on one point crossover can be found on Wikipedia[2].

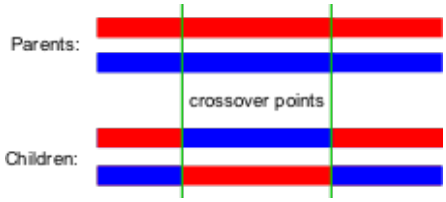

Figure 2: An example of two point crossover, obtained from `http://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)`

4.2.3 Uniform Order Crossover (UOX)

In UOX, each parent contributes approximately 50% of the genetic information of each child. This is done by generating a random bitstring of the same length as the parent chromosome arrays and using this as a mask. Alleles with a bit of one are obtained from the respective parent (a bit of 1 on parent 1 signifies to keep this allele in child 1, similarly for parent and child 2). Alleles with a bit of 0 are then obtained from the other parent, respectively. This allows the children to have a highly exploration-based variation of the parents. An example of UOX can be seen in Figure 3. More information on UOX is available on Wikipedia[2].

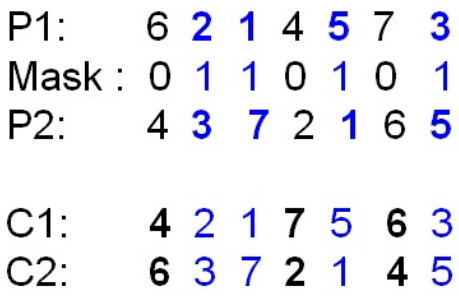

Figure 3: An example of UOX, obtained from lecture slides for COSC 3P71 by Dr. Ombuki-Berman.

4.2.4 Partially Mapped Crossover (PMX)

PMX is used for ordered chromosomes, that is we do not assume that any allele can be positioned in any location of the chromosome. In PMX, the genetic information is carried through to the children using 4 steps. The first step involved selecting arbitrary cut points in the parents. Then alleles from this range are then swapped in the children; information from parent 1 is copied to child 2, and vice versa for the other. We then copy the remaining information from parent 1 to child 1, where possible with no conflicts, and similarly for parent and child 2. The remaining step is to fill the remaining alleles with the remaining information that was not able to be copied in step 3. An example of PMX can be seen in Figure 4. More information on PMX is available on Wikipedia[2].

4.3 Mutation Operator

The mutation operator chosen for this application is based on inversion of a section of the alleles. There are 3 variations of this mutation, which are chosen randomly when mutation is applied. They are as follows:

1. Note inversion - inverts a section of the note frequency array.
2. Duration inversion - inverts a section of the note duration array.
3. Both arrays - inverts the same section in both the frequency and the duration array.

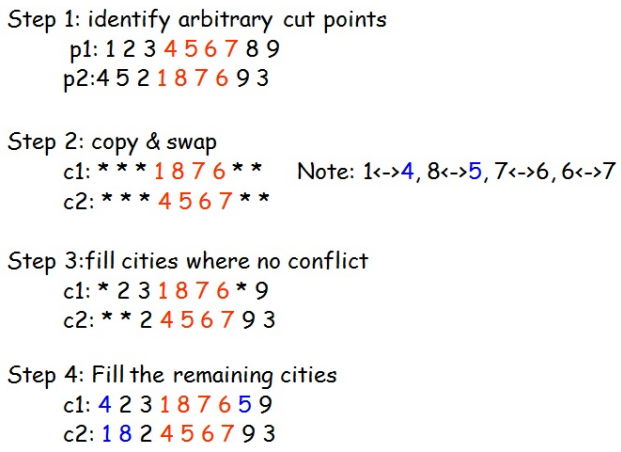

Figure 4: An example of PMX (applied to the traveling salesman problem), obtained from lecture slides for COSC 3P71 by Dr. Ombuki-Berman.

Each of these three methods uses the same formula. It takes approximately half of the array (starting at roughly 1/4th and ending at roughly 3/4th), and simply reverses this section. For example, an array with values [1,2,3,4,5,6,7,8] would become [1,2,6,5,4,3,7,8] after mutation.

4.4 BASS Library

This library was chosen as it has a very large API that is compatible with C#.NET 4.0, which is my language of choice. This library provides excellent support for audio input, output, stream management, sample management and many other features which were not used for the purposes of this application. The main component of the library that is used is the method for playback of a sample, represented with an integer array, through the default system audio device. This is done by creating a channel using the samples constructed using the chromosomes, and telling the library to interpret the information as audio data. The other function which is used is the WaveWriter which takes the sample array and writes it to a wave file.

The library is provided through a single dynamically linked library (DLL file)[1] which is quite small, and thus does not unnecessarily bloat the application. Since it was not originally intended to be for the .NET framework, a third-party DLL was also used to provide an API to these library functions.

5 How to Use the Application

This section will serve as a user manual to guide the user through using the application. Upon launching the application, the user will be presented with a form that handles the parameter input. These parameters will be explained below. Once the parameters are selected to the users preference, the "Generate Initial Samples" button, located on the top right of the form, should be clicked. This will generate 8 random genetic music samples which will be presented to the user on the bottom half of the screen, as

seen in Figure 5. Also, a reset button will now be shown, which will restart the GA by resetting the population.

The bottom left portion of the screen is where the samples are presented, as seen in Figure 5. Clicking any of the "Sample x" buttons will play the selected sample for the user to hear. Beside each sample is a numeric box which can hold values from 0 to 100. This is where the user is meant to enter the fitness of the corresponding sample. A higher value indicates a more pleasant sound.

Once the user has listened to each sample and selected a fitness for each sample, the "Perform Generation" button may be pressed. This will carry out one generation of the GA and create a new population using the specified crossover and mutation, possibly elitism as well. The 8 samples presented on the left will have changed now, and the fitness values will be reset to their default value of 50.

At any point while there is a population of samples created, a sample can be saved to a wave file. The button on the bottom right of the "Generated Samples" section, labelled "Save Sample as WAV" provides this functionality. The user must select the sample number in the box directly to the left of the button, possible values are 1 through 8. Clicking the save button will present the user with a save dialog box, which allows them to specify where the file will be saved, and what the file name should be. All files are appended with *.wav* as they are WAV audio files. Upon selecting this information, the file will be written to this location using the BASS API.

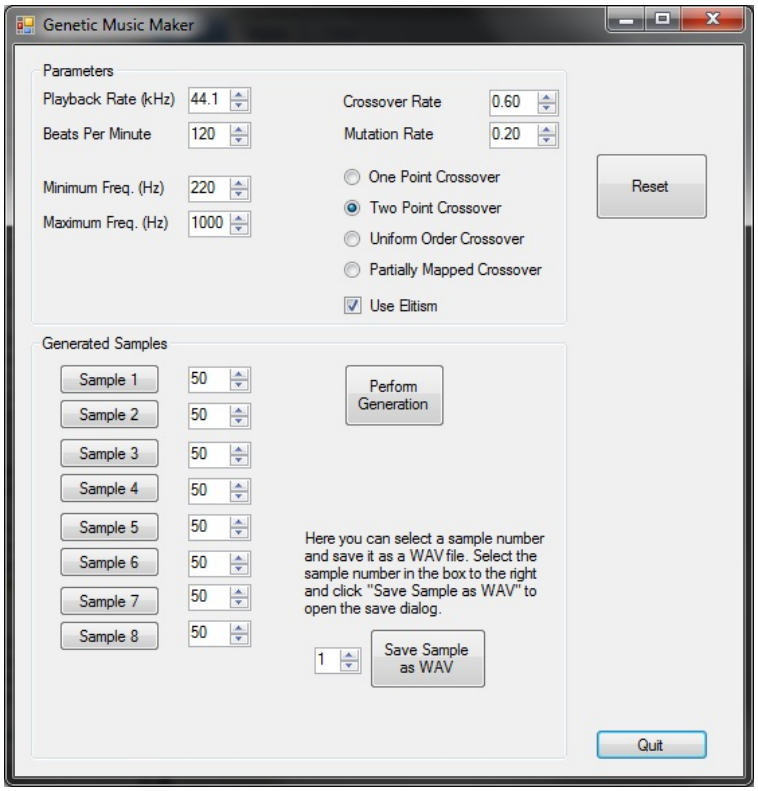

Figure 5: The main screen of the application, after the initial population has been created.

5.1 Running the application

No installation is required for this application, however in order to run the application, we need to ensure that .NET framework 4.0 is installed and that the correct libraries are in the current working directory. The 2 libraries that are needed are *bass.dll* and *Bass.Net.dll* and both of these must be present in the same directory as the executable in order to run the application as they are necessary for the playing and saving of the generated audio. Both of these libraries are present in the executable directories *"../bin/Debug/"* and *"../bin/Release/"*. The application can be run through Visual Studio, using the debugger, or as a standalone application by executing *../GeneticMusicGUI/bin/Release/GeneticMusicGUI.exe"*.

Please note that the BASS splash screen will always be present the first time the library is invoked (when playing or saving audio). This is due to the fact that it is not a registered copy, however it is freeware which is free to use in non-commercial applications with the only limitation being the splash screen.

5.2 Parameters

The top portion of the screen allows the user to select both audio and GA parameters for the application. The left column corresponds to the audio parameters and the right side to the GA parameters.

The audio parameters are explained here:

1. Playback Rate - the rate, in kilohertz (kHz), that the samples should be played back at.
2. Beats Per Minute - this controls the tempo of the generated samples. Higher BPM makes faster tempos.
3. Minimum Frequency - the minimum frequency value, in hertz (Hz), which tones can have.
4. Maximum Frequency - the maximum frequency value, in hertz (Hz), which tones can have.

The GA parameters are explained here:

1. Crossover Rate - the probability that 2 selected parents will perform crossover. If crossover is not performed, the individuals are simply copied to the new generation.
2. Mutation Rate - the probability that any given offspring will be mutated. The mutation is only applied to offspring generated through crossover, not copied chromosomes.
3. Crossover Type (radio buttons) - select the type of crossover that is to be used by the GA. More information is available in Section 4.2.
4. Use Elitism - if this checkbox is selected, single chromosome elitism will be used, that is the best individual will be kept each generation. If it is not selected, then the entire population will be replaced each generation.

References

[1] BASS and BASS.NET library downloads, as well as tutorials can be found here: `http://www.un4seen.com`.

[2] Information about UOX and PMX can both be found here: `http://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)`

[3] Various lecture slides from COSC 3P71, fall 2010 with Dr. Ombuki, were used for reference on GAs.