# Project Songbird

Rahim Sonawalla

ICS 175A

Fall: 12/06/05

## Table of Contents

## Introduction

If modern science is correct, then everything that surrounds us is the result of billions of years of constant change. Hemmed in emptiness, a spark of life in primordial soup started an unceasing series of birth, growth, reproduction, mutation, and death. To think all that we are today, all our knowledge, reasoning, and inventiveness can be traced back to a pool of slush on an empty planet is truly humbling. Our greatest ability is our creativity. The capacity to make something from nothing is remarkable. Music is one excellent example. A talented composer is capable of creating a musical piece that brings people to tears, anyone that has heard Pachelbel's Canon in D will agree. Project Songbird is an attempt to mimic biological evolution to create music.

Music is a very tricky subject to discuss, because it can become such a personal matter—what one person considers as beautiful music may be noise to another person. However, over time individual cultures have developed certain guidelines for creating music that is generally pleasing to the culture. It is important to keep in mind that these guidelines are not culturally independent; the concept of well composed music in India can be very different than the concept of well composed music in America. For this project, I have chosen to follow the norms of Western music, since we live in a Western society with primarily Western style music.

Norms for Western music are often referred to as Western Tonal Theory. Without going into too much detail, the basic principle of Western Tonal Theory is that all emotion is conveyed in the interval (the space) between two notes. Put into practice, melody lines that contain notes that stay within the key of the piece generally sound better, since the intervals between the melody and chords will be pleasing. Moreover, the sequence of certain notes within the key are more pleasing than other sequences. For example, the root note of the key (if the key was C, the root note would be a C as well) followed by the fifth note of the key (a G note if in the key of C) will sound better than the root note followed by a the second note of the key (a D note if, once again, in the key of C).

## Prior Research

Several researchers have used evolutionary algorithms to generate music. The most successful project is GenJam (Genetic Jammer) by Professor Al Biles, which is capable of taking in music and generating unique melodies to both accompany and solo in real-time. Researchers Marques, Olivera, Vierira, and Rosa have created a similar project which uses genetic algorithms to generate melodies without any other musical information (such as chords to play melodies over). Project Songbird builds on this existing research, but differs from it in that due to the limited amount of time, Songbird will not be in real-time—Professor Bines spent a year developing GenJam and 11 years refining it. Also, Songbird will be limited to generating melodies over a user specified key and chord progression, rather than being able generate melodies without any other information.

## Data Representation

To represent a melody, Songbird creates an "organism" object which holds, amongst other things, an array of "genes". Each gene holds information about a single note: the amount to rest before the note is played, the note to be played, and how long the note is to be played for. The gene data structure is represented in the figure below.

**Gene:**

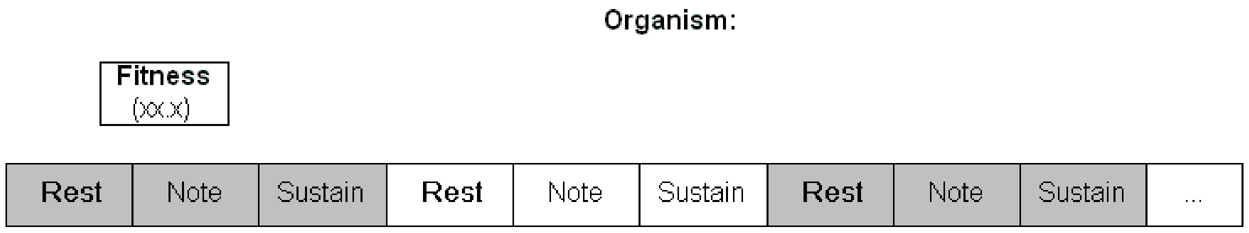
| | | |
|---|---|---|
| **Rest** | **Note** | **Sustain** |

| Rest values | |
|---|---|
| 0 | no rest |
| 1 | 1/32 note rest |
| 2 | 1/16 note rest |
| 3 | 1/8 note rest |
| 4 | 1/4 note rest |
| 5 | 1/2 note rest |
| 6 | whole note rest |

| Note values | |
|---|---|
| 48 | C 4th octave |
| 49 | C# 4th octave |
| 50 | D 4th octave |
| ... | ... |
| 81 | A 6th octave |
| 82 | A# 6th octave |
| 83 | B 6th octave |

| Sustain values | |
|---|---|
| 0 | no sustain |
| 1 | 1/32 note sustain |
| 2 | 1/16 note sustain |
| 3 | 1/8 note sustain |
| 4 | 1/4 note sustain |
| 5 | 1/2 note sustain |
| 6 | whole note sustain |

Along with genes, the organism also holds its fitness value (explained further in the next section). The organism data structure is represented in the figure below.



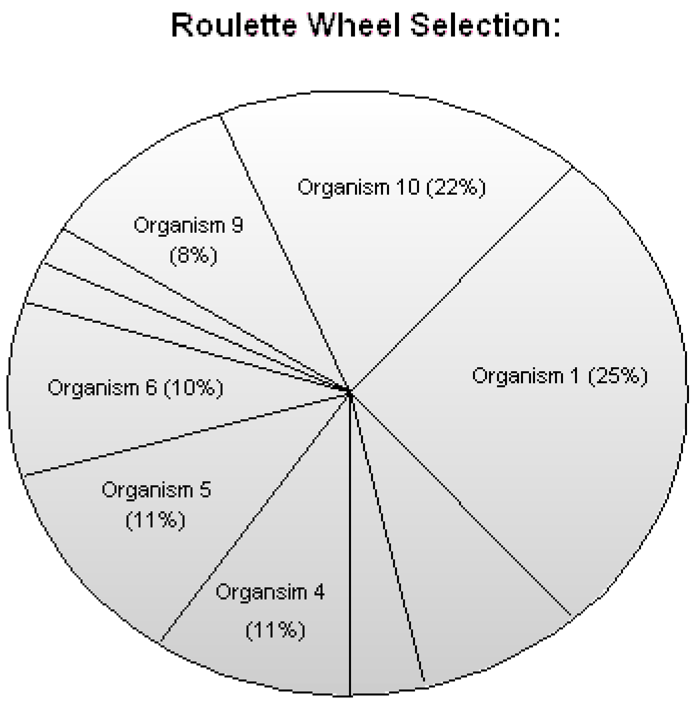
## Algorithm

Songbird uses a form of evolutionary computing called genetic algorithms. Genetic algorithms model biological evolution as described by Charles Darwin theory of evolution. Data structures act as “organisms” in a population, at each generation, each organism is evaluated for fitness similar to Darwin's survival of the fittest. Organisms with a higher fitness are more likely to be chosen to pass their genes on to the next generation. Some of these fit organisms will be chosen to “mate” with each other to produce offspring. These offspring will be produced by crossing genes from both parents. Lastly, some of these new children will be selected for mutation, where a random gene inside the organism will be mutated. This process will continue until a certain number of generations has passed, or a solution is found.

## Implementation

Genetic algorithms excel at finding a solution in a large search space. However, music is an infinite search space (since the combination of notes and the number of notes is infinite), so Songbird will stop its genetic algorithm after a user specified number of generations.

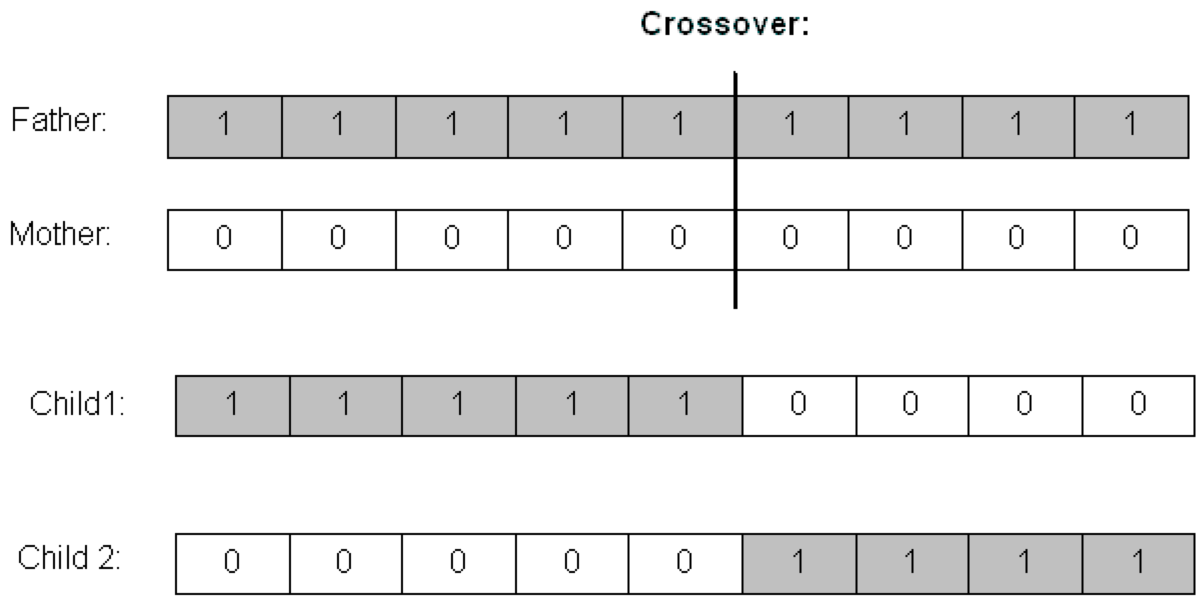
To select organisms to pass their genes on to the next generation, Songbird uses a roulette wheel selection process. To do this, Songbird takes the fitness of each organism in the population and assigns it a certain percent value that represents how likely the organism is to be selected. Each of these percent values can be thought of as sitting on a giant wheel, as the figure below illustrates.



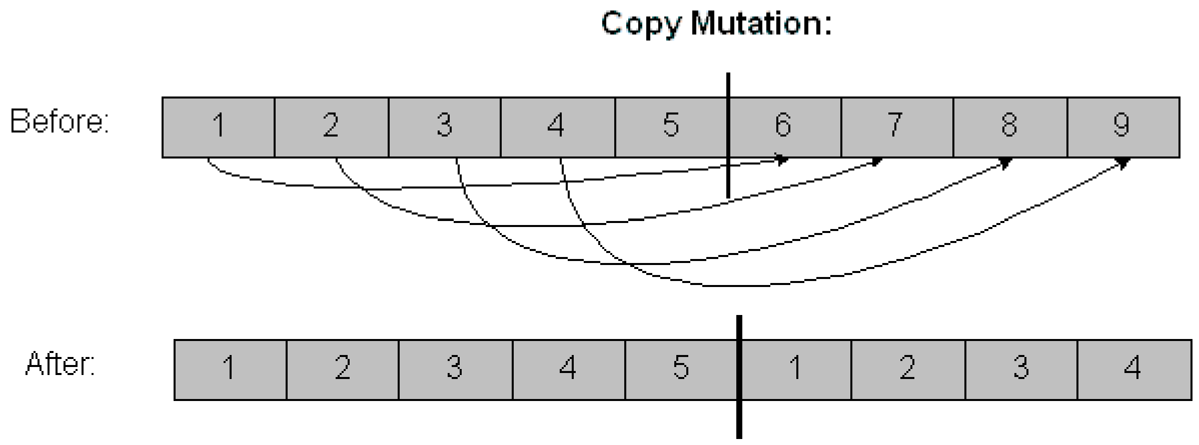
Once each organism in the population has been assigned a percent value, a random number is generated. If the random number is within the range of a certain organism, then that organism is selected to move on to the next generation. This process is then repeated until the number of selected organisms is equal to the population size. Naturally, it is possible that an organism will be selected more than once. Moreover, organisms with a higher fitness are more likely to survive, however, it is not impossible for an organism with a lower fitness score to survive. To help retain the best melody of the generation, the organism that generates the best melody is automatically selected to survive. This is known as elitism.

Crossover and mutation are not guaranteed to occur for every organism. This would be unfortunate, because it would cause the population to change rapidly and possibly cause it to lose many of the better solutions. Some crossover and mutation are needed, however, to ensure that the genetic algorithm does not get stuck around a local maxima while searching the space. The probability for crossover and mutation can be specified by the user so that he or she may see the effect of it on the genetic algorithm as the respective probabilities change.

To perform crossover, two organisms are chosen, since each organism is simply a collection of genes, a random point in the collection of genes is chosen. Every gene before that point in the father will make up half of the first child, and every gene after the chosen point in the mother will make up the other half of the child. This process is then repeated for the second child, however, every gene before the chosen point in the mother will make up the first half of the child and every gene after the point in the father will make up the second half of the second child. This process is known as single-point crossover and is shown in the figure below.



Mutation can take the form of one of four different mutations: gene mutation, note mutation, octave mutation, and copy mutation. In gene mutation, a random part of the gene (the rest value, note, or sustain value) is chosen and its value is randomly changed to another valid value. In note mutation, a random gene inside an organism is chosen and its note value is changed to a random note, however, its octave remains the same. In octave mutation, a random gene inside an organism is chosen and its octave is changed to a random octave, however, the note remains the same. Finally, in copy mutation a random point inside an organism is chosen, and every gene before the chosen point is copied to its corresponding position after the chosen point. This process is illustrated in the figure below.



To assign fitness to each organism, Songbird judges the melody stored in each organism based on notes, intervals, and note density. For every note that is in the key of the chord progression, the melody receives 2 points. For every note in the key that is a root note, the melody receives an additional 0.6 points. For every note in the key that is a third note or a fifth note, the melody receives an additional 0.5 points. For every combination of notes whose interval is not a tritone (an interval generally regarded in Western Tonal Theory as dissonant), the melody receives an additional 0.2 points. If, overall, the notes stay within two octaves, the melody receives an additional 5 points. If, overall, the melody has a distribution of notes where no one note makes up more than 50% of the notes in the melody, the melody receives an additional 5 points.

## Discussion/Evaluation

In retrospect, the project itself has been quite an evolution on its own. When I first started working on the project, I eagerly checked out several books on genetic algorithms from the Science library as well as reading whatever results Google would give me. After doing so much research on the topic, I felt confident that I would be able to complete Songbird and have it generate beautiful melodies for chord progressions. However, once I put away the books and sat down to begin the project, reality set in. Working on this project individually has given me the advantage of not having to deal with meeting times, communication issues, dealing with slackers, and managing shared code. However, being a one man team has had the disadvantage of having to do every part of the project—where other groups can lighten the load by splitting up the tasks, I must carry the entire weight. To help myself out, I tried to use many open source libraries to quickly develop the portions of the project that did not directly relate to the genetic algorithms.

I decided to start the project with the user interface so that I would have something to show for the first presentation. Being a C++/C# developer by profession, my only experience with Java were the times I had used it for other classes, such as ICS 21. I was disappointed to find that creating rich user interfaces in Java (something I had never done before) was an absolute pain. Rather than have an intuitive layout editor, I was forced to do it the archaic way and manually write out code to create basic widgets. Luckily, I was able to find a library to assist with coding the layout (Jgoodies), so once the widgets were created, layout and handling resizing were not a problem.

Not wanting my Songbird to be mute, I then went in search of documentation on how to play MIDI using Java. I was able to find a library that makes playing notes and chords through MIDI extremely simple (Jfugue). However, I did run into one problem with the library: the tempo for the output is not defined as beats per minute (the common way to express time in music), but rather the number of clock cycles a quarter note should receive (PPQ). Since the number of clock cycles is dependent on the type of processor, I was unable to make the conversion to the more common beats per minute (BPM). This may be confusing to users who are used to specifying tempo in BPM where a low BPM value implies a slower song, whereas a low PPQ value implies a faster song.

With the user interface and sound output taken care of, I began implementing the genetic algorithm. By meeting with both the professor and the TA, I was given some ideas on how to improve the success of the genetic algorithm by implementing a copy operator as well as copying the best organism over to the next generation. After implementing these suggestions along with the original algorithm described in my first presentation, I was sure the first run of Songbird would reward me with music. Like all software, the first time I ran it, I did not get what I expected, the program crashed with an “out of memory” error after just two generations. A day of debugging later, I managed to find the problem: an array was pointing to another array, rather than copying to the array, causing an infinite loop.

Upon fixing the problem and rerunning the program, I was happy to find that Songbird ran all the way through even with a large population and many generations, but was disappointed by the generated melodies. Very few of the notes were in the key, and the notes themselves jumped around octaves wildly. After spending a few days debugging and coming up short, I decided to look at all output generated by Songbird and found that the random numbers I had been using were far from random! Every organism in the population had the same melody, so the evolution was of course not very good. To fix this, I went in search of a better random number generator than the one included by Java. I found a library called RngPack which uses a lagged Fibonacci generator to generated research quality random numbers. After rewriting the project to use the new random number generator, the generated melodies were much better.

The best way to evaluate a program that generates melodies is to, of course, listen to the melodies. Clint Mansell, Robert Nyman, and all other modern composers need not lose any sleep tonight, the melodies generated by Songbird are not likely to put any of them out of a job. Although a vast improvement on the melodies generated by the prior faulty version of Songbird, the melodies generated by the newer version are not very interesting musically. This could be fixed in future versions of Songbird by expanding the evaluation function to look at the melody not just on a note basis, but also rhythmically. For the time being, I evaluate the success of Songbird on whether the melody generated by the final generation is an improvement on the melody created by the first generation. To do this, Songbird saves the melody generated by fittest organism in the first generation as well as the fitness of the melody. Moreover, the melody and fitness value are saved once again 1/3rd and 2/3rd of the total number of generations to run for to see if an improvement is being made. The output of a few sample runs is listed below.

**Results for:**

**Key**: C
**Population Size**: 10
**Max Generations**: 10

| *Generation* | *Fitness* |
|---|---|
| 0 | 54.00 |
| 3 | 57.90 |
| 9 | 62.70 |

**Results for:**
**Key**: C
**Population Size**: 100
**Max Generations**: 500

| *Generation* | *Fitness* |
|---|---|
| 0 | 60.90 |
| 166 | 59.50 |
| 499 | 72.80 |

**Results for:**
**Key**: C
**Population Size**: 1000
**Max Generations**: 500

| *Generation* | *Fitness* |
|---|---|
| 0 | 66.50 |
| 166 | 45.00 |
| 499 | 66.80 |

**Results for:**
**Key**: C
**Population Size**: 500
**Max Generations**: 1000

| *Generation* | *Fitness* |
|---|---|
| 0 | 66.50 |
| 333 | 45.00 |
| 999 | 71.80 |

Aside from a few generations where the fitness decreases (most likely the result of a mutation), the overall fitness of the melody improves as evolution continues. Although the resulting melodies are not very good, progress is clearly being made.

## References

- M. Marques, V. Oliveira, S. Vieira, AC Rosa. Music Composition using Genetic Evolutionary Algorithms.
- Miller, Willard. Evolutionary Algorithms Volume 111.
- Reeves, Colin R., Rowe, Jonathan. Genetic Algorithms Principles And Perspectives—A Guide to GA Theory.
- Wikipedia. Music. <http://en.wikibooks.org/wiki/Tonal_Music_Theory>.