EVOC: A Music Generating System using Genetic Algorithms

Timothy Weale and Jennifer Seitzer
University of Dayton
Dayton, Ohio, USA
{timothy, weale, Jennifer.seitzer} @notes. udayton.edu

Abstract

In this work, we present a Genetic Algorithm (GA) system "Evolutionary Counterpoint" (EVOC) that generates contrapuntal music. Counterpoint is the construction of a musical piece by superimposing multiple melodies, indirectly forming an underlying harmonic structure. Here, we include a description of the underlying algorithm, fitness function, and overall system modules.

1 Introduction

Counterpoint is the art of combining individual melodic voices to form a harmonious whole fGreenberg, 1997]. Contrapuntal music composition can be thought of as a vast search for the perfect combination of melodies. Given the exhaustive nature of the problem, genetic algorithms are ideal to facilitate the pursuit. In this work, we employ genetic algorithms to this end, and describe our implementation.

One aspect of counterpoint study involves taking a given musical line and creating an accompanying line. To do this correctly, the composer uses a set of guidelines (rules) for composition. These can be as simple as 'no dissonant sounds' to as complex as 'perfect intervals must be approached from opposite voices and one voice must step.' Using these sets of rules and a given melodic line known as a cantus firmus, one develops accompanying lines that adhere to every guideline. The guidelines are of utmost importance to our system because our fitness function is based on them. In our work, we have made the problem more manageable by using Species I counterpoint, the most structurally primitive form of counterpoint.

Sample Principles of Species I Counterpoint

1. Leaps of a third, fourth, fifth and ascending minor sixths arc available.
2. A distance of no greater than a sixth may be covered in one direction (with the exception of anoctave leap).
3. Only two leaps in one direction may be used consecutively.
4. Do not repeat notes consecutively.

Figure I

2 Music as a Constraint Problem

Constraint programming is concerned with using constraints to solve problems. Music composition of a counterpoint nature can be seen as a specialized constraint satisfaction problem (CSP). Any CSP has the following characteristics: a set of variables, a domain for the variables, and a set of constraints on the domain [Bartak, 1999]. Every note in a given cantus firmus line gives the composer at least one variable to satisfy. Every variable (note) is defined within the domain of every possible musical tone. Finally, the set of counterpoint guidelines gives us the constraints on the domain. The task of the composer (our system, EVOC) is to come up with a correct set of pitches along with its timing schedule. The version of counterpoint rules we are currently applying is limited to Species I in two voices (Figure 2), where given an upper melody as input, the system composes the bass line. We employ the counterpoint rules found in a typical music theory setting [Magnuson] some of which are shown in Figure 1.

3 Main Algorithm

```
begin EVOC-Algorithm( )
  input: cantus firmus line
  output: Species I melody line

  Randomly generate 8 initial melodies (parents)
  repeat
     1. create 32 offspring using crossover on parents
     2. randomly mutate offspring
     3. apply fitness function to offspring
     4. select new parents from offspring
  until fitness function returns 0
```

Cantus Firmus and Counterpoint Line Using Species I in Two Voices

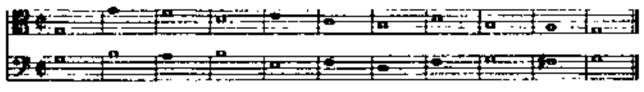

Figure 2

4 System EVOC

The EVOC system consists of two independent program modules each invoked by the Visual Basic interface. Module One reads in the input MIDI file and transforms it into an intermediate text format. Module Two uses this to run the genetic algorithm and generate a counterpoint line. A graphical representation of the logical organization can be found in Figure 3.

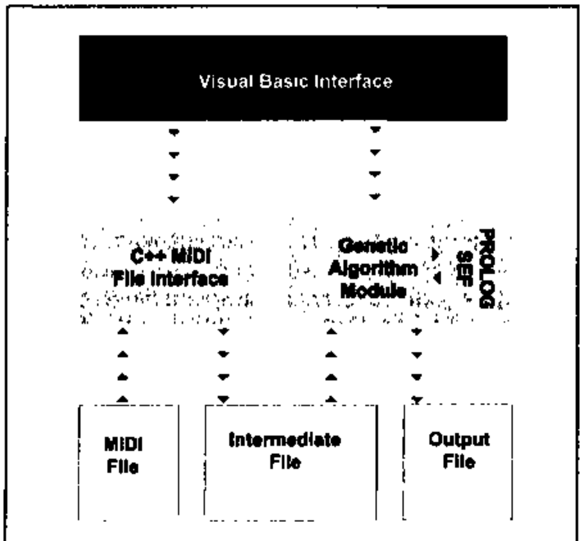

Figure 3

The Visual Basic interface provides a simple, easy way to invoke the other two modules. After loading the interface, the user defines the name of the MIDI file with the given melody line. Using this file name, the additional two modules are invoked.

The MIDI input module is a C++ program that reads in a MIDI file and outputs it as an intermediate file. The melody (as represented by a MIDI file track) is read from the file using file-processing routines implemented in Sapp's IMPROV suite 12002]. From there, the lines are converted to an internal representation, from which the intermediate file is created.

The final module runs the genetic algorithm. This part reads the intermediate file, initializes the algorithm, and employs the main algorithm detailed in Section 3. This algorithm iterates until the fitness function returns zero indicating that a correct melody has been generated. The output of this final module is the generated contrapuntal melody.

4.1 PROLOG Fitness Function

In any GA system, the fitness function indicates which members of a generation survive. For our fitness function, we wrote a theorem prover that embodies the historical Species I counterpoint guidelines. By comparing any given genotype melody with the guidelines, we assign a valuation of validity to the genotype. During selection, the highest rated melodies persist to the next generation.

The evaluation function first constructs a list of *note()* values. These notes have the form: *note(MIDI_value, MIDI_name, measure).* With these note lists, the PROLOG evaluation is done using multiple scans, with two primary areas of investigation: melodic adherence and harmonic adherence. Compositional errors occur when the constraints of the program (the species counterpoint rules) are broken. When such an error is encountered, an appropriate penalty is assessed against the particular genotype valuation. Serious infractions (e.g., parallel fifths) suffer a -500 penalty, while non-serious violations (e.g., counterbalance issues) may only suffer a -100 penalty.

5 Conclusion

To date, we have an operational system that can generate valid counterpoint lines in Species I. The output is correct and perfectly valid as a functional counterpoint line. We are currently working on improving the rule base as well as the user interface for EVOC.

In this poster, we presented a genetic algorithm and a theorem proving deductive fitness function for the searching and generation of musical counterpoint. It employs a theorem prover as a fitness function that houses fundamental guidelines used by musicians composing these pieces.

References

[Bartak, 1999] Bartak, Roman. Constraint Programming: In Pursuit of the Holy Grail. Proceedings of Week of Doctoral Students (WDS99), Part IV, MatFyz Press, Prague, June 1999, pp. 555-564

[Greenberg, 1997] Bernard S. Greenberg; Bach FAQ Site, 1997 http://www.bachfaq.org/ctpt.html. (date verified i 1/1/2002)

[Magnuson] Magnuson, Phillip. *Species Counterpoint.* http://maestro.udayton.edu.

[Sapp, 2002] Sapp, Craig S. *Improv.* http://improv.sapp.org.

lSchottstaedt, 1989] Schottstaedt, W. Automatic Counterpoint. Current Directions in Computer Music. MIT Press, Cambridge.