Fugue Composition with Counterpoint Melody Generation Using Genetic Algorithms

Andres Garay Acevedo

Georgetown University – Washington DC, USA
ag66@georgetown.edu

Abstract. This paper presents the results of implementing and evaluating a genetic algorithm to assist in the task of automatic counterpoint generation. In particular, a fugue subject was used as an input for the system, while the generated counterpoint melody was to act as the countersubject. The genetic algorithm was tested with two different input melodies, and a basic set of rules for fitness evaluation. Within this domain, the results were satisfactory. Finally, the suitability of genetic algorithms for the task of rule-based melody generation, as well as possible future work and enhancements to the implemented system, are also reviewed and discussed.

1. Introduction

The task of composing music with the assistance of a well-defined algorithm has been a greatly debated over time. W. A. Mozart's dice game showed the feasibility of generating simple pieces with an aesthetical appeal by using nothing more than a set of musical motives and a pair of dice. Recently, the debate has focused on the use of computers to create music by obeying a strict set of rules defined by the composer [1].

Leaving the debate aside, one can see the task of composing music as that of applying formal rules over a starting melody until a satisfactory result is achieved. Mathematical relations, for example, are one of the important rules used during this process of incremental revision. The resulting piece is then a product of what Bruce Jacob defines as a "hard work composition" [9], a composition that doesn't come from a magical moment of inspiration but rather from hours of frustrating an arduous methodical work. Other authors use the term "compositional loop" [11].

One can then imagine a computer as a form of help in this process of *hard work* composing. The role of the composer isn't replaced by the computer, which just acts as an assistant that carries out part of the mentioned *arduous work*, thus leaving the composer with time to focus on other aspects of the composition. The task becomes then to identify the specific processes that are well suited to be carried by a computer.

In this paper we explain why counterpoint generation is such a process. Then we present the results of implementing a small-scale system that uses a genetic algorithm in order to find a suitable counterpoint melody that can be arranged in a basic fugue structure. Finally, some observations and possible future work are discussed.

2. The Problem

2.1 Motivation

Since William Schottstaedt proposed a system for automatic counterpoint melody generation [12], no other author has approached the problem by using a search algorithm. Instead, stochastic models have been primarily used instead for melody generation [13,8], while other authors use genetic algorithms to focus on higher aspects of the compositional loop [11].

Given this situation, it is valid to ask if genetic algorithms are a valid technique for solving the counterpoint melody search problem. Schottstaedt's system [12] used a best-first search algorithm to find a suitable solution for the problem. We, in turn, are interested in evaluating the suitability of genetic algorithms for this task. In particular, we are interested on counterpoint melody generation for the task of arranging a simple fugue. This is a two step process that is now presented.

2.2 Problem Definition

"For several hundred years composers have praised species counterpoint as one of the best ways to learn to write music" [12, pg. 199]. Some of the formal rules for a *good* counterpoint have even been stated since the eighteenth century; for example, J.J. Fux's "Gradus ad Parnassum", published in 1725, presents a compilation of these. Such a well studied process starts to suit the idea of computer assisted composition. In more general terms, the process of generating valuable musical phrases, themes or whole pieces can be stated as a computational task. For the particular case of counterpoint melody generation, the task can be formulated as a search problem: Given an initial musical phrase, find a suitable counterpoint melody from the space of all possible melodies.

However, this proves to be an NP-complete problem once the search space is formalized [12]. The search domain is essentially unlimited given the combinatorial possibilities of individual notes in time, rhythm, harmony and melody [3]. If there are 16 ways of *moving* from one note to the next, then a short musical phrase of 10 notes will generate a search space of 16^{10} possible solutions. Given this situation, it becomes necessary to impose some constraints to the search space (or to the decision making algorithm) in order to make a proposed solution feasible.

Once the problem of finding an appropriate counterpoint melody has been overcome, the compositional process can continue. Specifically, in this project the idea was to generate a set of valid counterpoint melodies and arrange them in order to create a Fugue. This encompasses the creation of an answer and two counter-subject melodies, given the initial subject of the fugue. The musical phrases are then arranged over time for three voices as shown in table 1.

The process of reviewing the counter-subjects and arranging them in the basic fugue structure can be performed both by a human composer as well as by a computer. If an algorithm is to be used, then it must incorporate higher-level concepts of musical arrangement and coherence; it must have some sort of aesthetic value. This constitutes a different problem than that of creating melodies; nonetheless, it is a task that has also been widely studied and described, as will be seen in the following section. For the purposes of this paper, we assume that a human performs this arrangement, based on his personal aesthetics.

Table 1. Basic fugue structure

VOICE	PART 1	PART 2	PART 3
Soprano	Subject	Counter-Subject 1	Counter-Subject 2
Alto		Answer	Counter-Subject 1
Bass			Subject

3. Literature Survey

As can be expected, researchers have focused on the specifics of music composition described here. While some study the task of melodic phrase generation, others do so for issues of musical arrangement. Nonetheless, systems that combine both tasks have been developed and proven to be of great utility.

3.1 Melody Generation

Most popular systems have focused on generating notes based on stochastic methods, where each note is produced based on its conditional probability, given its preceding notes [13,8]. Such probabilities could be modified so that a certain "style" is encouraged.

Less rigid systems that rely on nonlinear dynamical systems have also been implemented. These are described as systems of mathematical equations, and display a behavior found in a large number of systems in nature [1]. The way in which notes are calculated is through a process of iteration, where the solution to the equations is calculated and then fed back into the system as an input value for the following iteration. The generated solutions can be viewed as a set of n points in space; this set is known as an orbit. Of special interest among composers are chaotic orbits, which display semi-cyclical behavior. That is, the musical melodies creates seem to "wonder around" a musical motif but without repeating the same melody.

The systems mentioned above generate melodies from scratch; at most they have mathematical functions as input. Another approach to the problem consists of generating the resulting melodies based on a given musical phrase, which acts as an input. Probably the most successful implementation of such an approach is GenJam [2], a system that uses genetic algorithms in order to *improvise* over Jazz melodies.

For these systems, the quality of the produced musical phrase must not be judged by itself. Rather, the melodic content of a reference phrase is used for fair evaluation. Applications of these systems include melodic development, thematic bridging, and harmonization. Burton & Vladimirova [3] give a compilation of musical applications were genetic algorithms in particular have been used for melody generation.

The specific problem of counterpoint *solving*[1] has been formally stated as a search problem that aims to use the formal rules of counterpoint composition that have been defined over time [12]. Nonetheless, further development of the problem or alternative solutions have not been published lately.

3.2 Arrangement

Artificial Intelligence techniques have been widely used to model compositional tasks of *higher level*. Most of the systems seem to use a generator/modifier/selector approach that was first proposed by Lejaren Hiller during the 1950's [6]. In this approach, the output of a musical generator (such as the ones described above) is fed into the modifier and selector subsystem, which performs melodic development over the main theme and then arranges the melodies in a suitable way. Jacob calls these components composer, ear and arranger respectively [8].

The arranger subsystem can be implemented in different ways. It can be a complex rule-based system, such as David Cope's EMI [4], which produces a deterministic behavior. It can also be a looser system that, for example, just organizes the melodies so that they follow a given contour pattern.

Bruce Jacob's *arranger* [8] implements another approach that is widely used in algorithmic composition: interaction between the system and a human composer. Jacob's system uses a Genetic Algorithm that selects possible phrase candidates and then presents them to the user. Once the user selects the ones it considers most fit, the algorithm extracts their parameters and employs them in future iterations.

Moroni's Vox Populi [11] goes even further as this interaction occurs in real time. The human composer makes use of a graphical interface, where can adjust the parameters of the Genetic Algorithm as the composition is performed. Alternatively, he can draw two curves that represent which are represented as variables that guide the compositional process.

The particular problem of arranging a simple fugue is discussed by Milkie & Chestnut [10], although it can be said that their project consists of an instantiation of the more general framework developed by Lejaren Hiller. However, their use of genetic algorithms is limited only to the generation of the fugue's subject. In this respect, the work resented here can be considered both different as well as more general.

4. Implementation

A system that generates a counterpoint melody was implemented and evaluated. The system architecture is depicted in figure 1. This system uses a simple genetic algorithm (SGA) in order to generate a melodic phrase, given a referent melody. In particular, the input melody can be seen as the fugue subject, and the output melody as a counter-subject.

[1] Some of the literature refers to this task as species counterpoint.

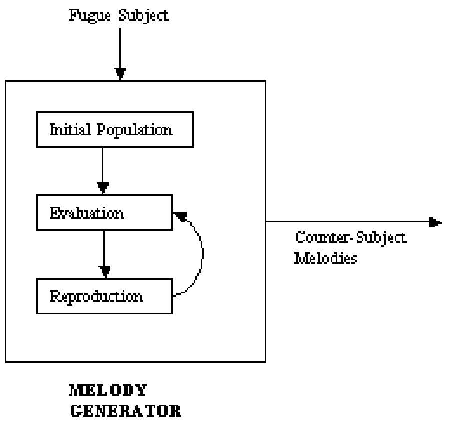

Fig. 1. System overview

When implementing a Genetic Algorithm (GA) based system it must be remembered that the system’s success is dependent upon three characteristics: Search domain, fitness evaluation and input representation [3]. The search domain was specified in the previous section. In turn, how the latter elements were implemented is now discussed.

4.1 Melody and Note Representation

Melodies and notes were implemented as two different objects, using an object oriented programming language (OOP). Both representations are now discussed. First, however, it must be stated that the representation of a note is very similar to the one used by Biles [2] in GenJam, with the idea of a feature field borrowed from Milkie & Chestnut [10]. Similarly, the representation of a melody was borrowed from the latter.

Notes are characterized by three different attributes: pitch, length and feature. In the implementation, the *pitch* corresponds to an integer that represents the number of semitones between of the first note of the melody scale and the represented note. For example, if the note has a pitch value of 2 and the scale is C major, then the encoded note is a D. In particular, as notes were limited to a single octave for the experiment, the pitch value ranged between 0 and 11.

The *length* of each note was encoded as an integer, ranging from 0 to 3. A simplification was made here, as only 4 different lengths could be represented. This decision was made because we wanted melodies to have a quaver as the shortest note, given that this was the shortest note found on our input test set. Thus, the note durations encoded were: half a beat (0), one beat (1), two beats (2) and four beats (3).

Finally, the *feature* field was used to store extra information about a note. A value of zero was used to represent a simple note, a value of 1 to represent a silence or rest (thus eliminating the meaning of the pitch value), and a value of 3 to represent a hold.

Melodies, in turn, have five different parameters. The first one is the *key* of the melody. This was encoded using an integer value. In particular, we used the value 0 to represent a key of C major, the one used for all the melodies of our test. The second parameter is the *beat* of the melody. Here we also used an integer, and represented a beat of 4/4 with the value 0 for our test.

The melody *length*, in number of notes, was also stored as an integer value. The fourth parameter corresponds to the melody *fitness*, which was an attribute used specifically by the genetic algorithm. Finally, an array containing the actual notes is the last element that conforms a melody.

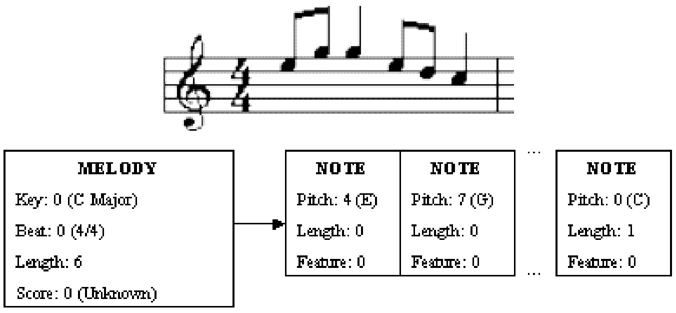

Fig. 2. A melody fragment and its representation

In order to better illustrate what has just been explained, a fragment of a melody and its representation are presented in figure 2. For each attribute, the store (encoded) value is shown after its name. The represented value is shown in parenthesis.

4.2 Fitness Evaluation and Crossover

The second element crucial to the performance of the genetic algorithm is the fitness evaluation. After carefully reviewing the literature, we decided to use the following function for the implemented system:

$$fitness = \sum_i weight_i * feature_i$$

where a set of features are evaluated and multiplied against a preset weight. This scheme is very similar to Schottstaedt's one of *prohibitions* and *penalties* [12], as some features were derived from his work. However, in order to adapt the rules for the genetic algorithm, melodies are rewarded when they display a certain feature, instead of being penalized for containing a *prohibition*. The set of selected features and its corresponding weights is presented in table 2.

Table 2. Selected features for fitness evaluation

Feature	Weight
Percentage of notes in the same key as the input melody	100
Same length in measures as the input melody	100
Melody ends in a consonant note	50
Melody begins with a consonant note	50
Percentage of intervals smaller than a third	100
Number of repetitions (higher than 2) of a note	-25

Once the fitness was calculated for each member of a generation, a single point crossover operator was used to create the offspring of a population. The crossover point was selected at random and corresponded to a particular measure at which the melodies are divided and then recombined to form two new melodic phrases. Figure 3 shows two sample melodies, and the resulting descendents when they are combined at the first measure. Note that if the value of zero is chosen as the crossover point, then the two parent melodies will be carried to the next generation.

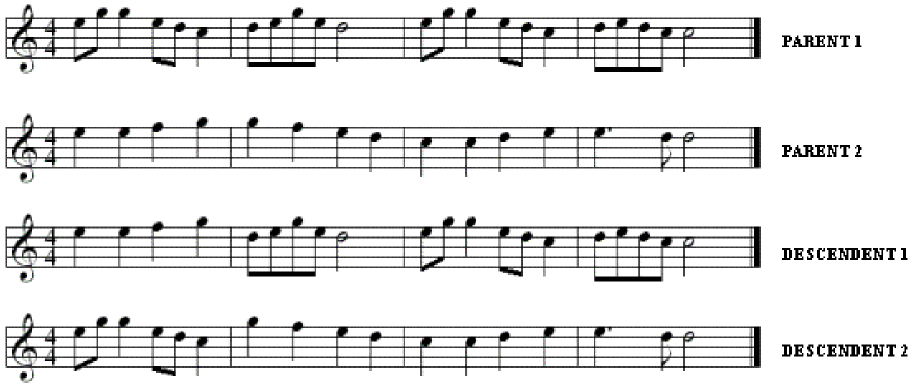

Fig. 3. Crossover at measure 1

By combining at a given measure instead that at a given note (i.e. at the third note), we could guarantee that the resulting melodies would have the desired length. This, in turn, guaranteed a minimum fitness on the newly generated population.

Finally, it is important for a genetic algorithm to have a mutation operator in order to assure that a suboptimal solution is found [5]. A mutation is no more than an evolution operation performed on an individual of a population. In particular for our case of musical phrases, mutation consists of a set of musical operations that are be performed on the original melody. The particular operations consist of transposition and inversion of a single note, randomly selected. These operators were selected to manipulate the content of the system and evolve the population in a musically meaningful way [3].

5. Measuring Performance

A test was carried to measure the quality of the melodies produced by the described system. In this section we describe how the test was performed, and then present the results found. Finally, these results are discussed.

5.1 Test Results

The genetic algorithm described in the previous section was run several times, using two different input melodies. Both of the input melodies were four measures long, with a beat of 4/4, and in the key of C major. This means that the output of the system was expected to present this same set of characteristics. Given the data representation selected, the fitness function and the crossover operator, all resulting melodies displayed these characteristics.

For the genetic algorithm, a population size of 50 individuals was used, and 100 generations were produced. These values were determined experimentally, and provided the best average fitness of the population.

Once the output melodies were collected, they were fed as the input of a MIDI sequencer. After this, they were played alongside a faded version of the input melody used for generating them. A sample melody, generated during the test, along with the input used for creating it are shown in figure 4.

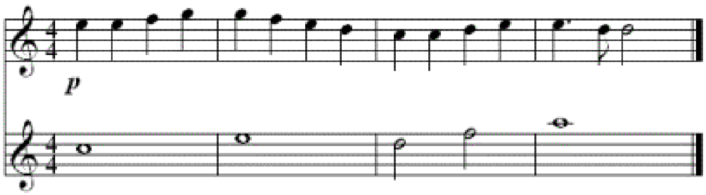

Fig. 4. Sample melody generated by the genetic algorithm

After collecting the results, the set of selected melodies was presented to an expert, who was then asked to grade them according to how well the generated melody could act as a fugue counter-subject for the input melody. That is, the generated melody was not evaluated by itself, but with respect to the system's input. A scale of 1 to 5 was used for this evaluation, were a value of 5 corresponded to the output melody being a good counter-subject for the fugue's main subject. A value of 1 represented no melodic relation between the two melodies.

The average grade given by the expert was 2.94. In addition, table 3 summarizes the percentage of melodies that were given each one of the five possible grades.

Table 3. Summary of results

GRADE	Percentage of Melodies
1	11.1%
2	16.7%
3	38.9%
4	33.3%
5	0%

Even though these figures might seem somewhat low at first sight, it must be stated that a grade of 5 corresponded to the melody being the work a musical expert (human or machine).

One final consideration that must be taken into account when evaluating the results of a melody generation system is that the generating algorithm must not converge upon a single solution [11]. Rather the solutions must display a set of recognizable characteristics without being a mere repletion of each other. There is not just one possible solution to the task of counterpoint generation.

The system tested displayed this desired behavior. In particular, no two melodies produced during the test are identical. This can be explained in part because the genetic algorithm was run for a fixed number of generations, and the algorithm doesn't converge upon a set of solutions at a fixed rate. That is, the average fitness of the final population is not necessarily the same for two repetitions of the same experiment.

This is a particularly interesting result, as this characteristic might not always holds true for counterpoint-solving algorithms based on other search techniques. Given this, genetic algorithms prove to be a viable technique for solving Schottstaedt's original formulation of the problem. Moreover, they might be a more suitable technique than other search algorithms, not only from a technical standpoint but also from a musical one. However, further testing and comparison must be done before making such a powerful claim.

5.2 Discussion

The results presented here are encouraging and promising. However, the system is far from being perfect; moreover, it's performance is not good enough to satisfy the needs of an average composer[2]. Further work and testing are both needed and encouraged. In anticipation of this, in this section we present a discussion of what we think are some of the necessary steps that must be taken in order to improve the results.

The first step needed in order to achieve better results consists of using a more elaborated fitness function for the genetic algorithm. By this we mean that a more complete set of features must be developed. Schottstaedt's set of rules (or penalties) provides a thorough compilation of the most important counterpoint rules, along with a measure of relative importance (their penalty). Other sources might be found in the musical literature.

This task shouldn't be very difficult, as it is highly localized; however it is important to state here another related problem, namely, that of data representation. The melody representation chosen is very useful and meaningful; nonetheless, the algorithm might benefit from a more detailed description.

Here a distinction must be made. The task of augmenting the set of features used by the algorithm relates directly to the problem at a higher level. On the other hand, that of using a more detailed data representation relates to the particular implementation. Nonetheless, with this distinction in mind we still find it useful to talk about the issue of data representation.

In particular, we think that some important data is lost when a melody is represented as a set of notes, grouped under a set of common characteristics. A more detailed description should make explicit the concepts of both measure and beat. Once these concepts are explicit, the set of operations that are performed on the population (such as crossover and mutation) will be much more efficient. Moreover,

[2] Species counterpoint exercises are not expected to produce great music.

implementing a more complete fitness evaluation function should become an easier task. This explains the importance of mentioning this particular implementation issue.

Our final recommendation pertains to the task of generating the initial population of the genetic algorithm. This task, we see, can be carried in two different ways. For our test system we decided to create the initial population of musical phrases using a random number generator, in accordance with most implementations of SGAs. In this way, the search space isn't constrained.

However, the initial population can also be created by using a stochastic note generator, such as the one proposed by Jacob [8]. This imposes a constraint on the search space; but that, in particular, might be a desired effect. By using a stochastic model for the creation of the initial population, a minimum level of fitness (and thus quality of the system) can be guaranteed. Moreover, some of the features that we used in our fitness evaluation can be eliminated. In particular, those that assess only to the individual being evaluated. If these features are removed, then the evaluation function will be comprised of a set of rules that refer only to the relation between the given individual and the referent melody. This can be then viewed as a cleaner evaluation, as it concentrates only on counterpoint specific features, rather than on those that relate both to the counterpoint task and the structural quality of the individual. However, it must be clear that what has just been described is a tradeoff between the clearness of the problem definition and modeling, and the completeness of the search space. It should be clear by this point that we decided to have a complete search space.

6. Conclusion

A basic algorithmic composition system has been proposed. The system finds a suitable set of melodies to be arranged in a fugue, given the fugue's main subject. Particular attention and effort has been put into the design of the melody generator, which uses a genetic algorithm in order to evolve generations of candidate solutions for the problem.

A test was carried to measure the quality of the results produced by the described system, and the results were discussed. In particular, we were very pleased with the results produced by one of the two test sets. Nonetheless, the overall results are also encouraging.

Future work and some specific issues were discussed. Some of these issues relate to the particular implementation of the system, while others relate to the problem specification. Both are considered important. Accordingly, future work is strongly encouraged.

Finally, it must be stated that the use of genetic algorithms has proven to be a viable technique for solving the problem of automatic counterpoint melody generation. Moreover, given some characteristics of the problem, such as the size of the search space and the existence of multiple solutions within this space, the use of a heuristic search method, and genetic algorithms in particular, seems to be a well suited mechanism for achieving a satisfactory solution.

References

1. Alpern, A. 1995. "Techniques for Algorithmic Composition of Music." On the web: http://hamp.hampshire.edu/~adaF92/algocomp/algocomp95.html
2. Biles, J. 1994. "GenJam: A Genetic Algorithm for Generating Jazz Solos." *Proceedings of the 1994 International Computer Music Conference*. San Francisco: International Computer Music Association.
3. Burton, A., and Vladimirova, T. 1999. "Generation of Musical Sequences with Genetic Techniques." *Computer Music Journal* 23(4): 59-73.
4. Cope, D. 1992. "Computer Modeling of Musical Intelligence in EMI." *Computer Music Journal* 16(2): 69-83.
5. Goldberg, D. 2000. "The Design of Innovation: Lessons from Genetic Algorithms, Lessons for the Real World." *Technological Forecasting and Social Change* 64: 7-12.
6. Hiller, L. 1981. "Composing with computers: A progress report." *Computer Music Journal* 5(4): 7-21.
7. Horner, A. and Goldberg, D. 1991. "Genetic Algorithms and Computer-Assisted Music Composition." *Proceedings of the 1991 International Conference on Genetic Algorithms*. San Mateo: International Society for Genetic Algorithms.
8. Jacob, B. 1995. "Composing with Genetic Algorithms." *Proceedings of the 1995 International Computer Music Conference*. San Francisco: International Computer Music Association.
9. Jacob, B. 1996. "Algorithmic Composition as a Model of Creativity." Organised Sound 1(3).
10. Milkie, E., and Chestnut J. 2001. "Fugue Generation with Genetic Algorithms". On the web: http://www.cs.cornell.edu/boom/2001/Milkie
11. Moroni, A. et al. 2000. "Vox Populi: An Interactive Evolutionary System for Algorithmic Music Composition." *Leonardo Music Journal* 10: 49-54.
12. Schottstaedt, W. 1989. "Automatic Counterpoint." *Current Directions in Computer Music*, pp. 199-213. Cambridge, Massachusetts: MIT Press.
13. Supper, M. 2001. "A Few Remarks on Algorithmic Composition." *Computer Music Journal* 25(1): 48-53.
14. Temperley D., and Sleator D. 1999. "Modeling Meter and Harmony: A Preference-Rule Approach." *Computer Music Journal* 23(1): 10-27.