

# Melodic Composition with Genetic Algorithms

Michael J. Korman

May 5, 2004

## 1 Introduction

Performing creative activity with artificial intelligence is a highly controversial topic. Many claim that this is a uniquely human activity that can never be emulated by algorithmic means. Others stand by the belief that any human process is duplicable through a clear understanding of the mechanisms that humans employ in producing the behavior. In this project, we attempt to explore the issue by attacking the specific problem domain of music composition.

## 2 Genetic Algorithms

Genetic algorithms[2] are a method of discovering optimal solutions in a large search space by simulating evolutionary progression. To understand GAs, we must first review concepts from biological evolution.

Practically every living organism contains its physical description encoded in *DNA*. The DNA is composed of a set of *chromosomes*, each of which is made up of a certain number of *genes*. Every gene in the chromosome is responsible for a specific trait of the organism.

Evolution is controlled by two processes: *mutation* and *crossover*. Mutation occurs when a gene on a chromosome randomly changes value. This introduces diversity into a population. Evolution cannot happen without mutation, since lack of mutation will lead to lack of change, and hence lack of improvement. Crossover occurs in organisms that reproduce sexually. During reproduction, the chromosomes of the parents split in half and are reconstituted in the new organism by combining the first half of one parent's chromosome with the second half of the corresponding chromosome in the other parent. This leads to a faster rate of improvement in the population, since once a trait has been mutated into existence, it can be propagated throughout the population.

Genetic algorithms use this idea to optimize search problems. Suppose we want to find assignments of variables that maximize certain characteristics. Each variable will be represented by a gene. The *quality* of a given chromosome takes on the quality of that combination of features.

We must now define a *fitness metric* that tells us how likely it is that a specific chromosome will survive. The *standard fitness metric* is often used for

this purpose. Letting  $n$  be the number of chromosomes,  $f_i$  the fitness of the  $i$ th chromosome, and  $q_i$  the quality of the  $i$ th chromosome, we define the standard fitness as follows:

$$f_i = \frac{q_i}{\sum_{k=1}^n q_k}. \quad (1)$$

The genetic algorithm can be performed in six basic steps:

1. Begin with a few starting chromosomes, basically chosen at random.
2. Sort the chromosomes by fitness, and choose the highest rated to act as parents of the next generation.
3. Perform crossover by mating the selected chromosomes.
4. Apply mutations at random.
5. Kill off chromosomes who have fitness below a specified threshold.
6. Add the new chromosomes to the population.

Assuming that we chose our fitness metric properly, this should continue until the desired feature is maximized.

### 3 Implementation

In the chosen knowledge representation, chromosomes contain descriptions of some measures of music. Each measure is a sequence of notes, and each note contains both pitch (rests have a special pitch of 0) and duration information. Durations are limited between 1 measure and 1/16 measure. To achieve metric consistency, we must enforce the constraint that the durations of the notes and rests in a measure add up to 1. Additionally, pitch is limited to the following interval:



This allows for a total of 24 distinct pitches.

Crossover occurs at a single random point on two chromosomes.

When a chromosome is mutated, one of the following operators is applied to it:

**Transpose** - Add 5, 7, or 12 to all pitches. In the example below, each pitch is increased by 5. The first staff shows the original, and the second shows the modified version.



**Sort** - Sort ascending or descending. The staves below are sorted in the ascending order. Note that pitches are moved up and down by octave to ensure that the notes are sorted.



**Permute** - Reorder notes. This allows notes to be reordered, along with their pitches and durations.



**Invert** - Permute pitches. Invert causes the pitches of notes to be permuted, while the durations remain constant.





**Re-meter** - Randomly change durations. In this case, durations in each measure are changed randomly, while pitches remain constant. This can allow the number of notes in a measure to change, but metric consistency must be preserved.



**Perturb** - Change the pitch and duration of a random number of notes, preserving metric consistency. This is equivalent to a completely random mutation.



At each step, the resulting chromosome is evaluated according to the following characteristics:

- Jumps** - We assign a lower score if there are large intervals between notes.
- Repetition** - We assign a higher score if the chromosome contains repeated patterns.
- Liveness** - We assign a higher score if the chromosome contains more short notes than long ones.
- Pitch Variance** - We assign a lower score if there are a lot of identical consecutive notes.

**Rests** - We assign a lower score if there are a lot of rests.

The algorithm works by applying the procedure detailed in §2. It begins with an initial population consisting of ten copies of a *seed* input, specified by the user. At each generation, the chromosomes are evaluated by the metric in Equation 1.

The user has the option of specifying the number of generations, the mutation rate, probabilities of each of the mutation operators, relative importance of each evaluation characteristic, and the threshold used in killing unfit chromosomes.

The genetic algorithm was written as a command-line C++ program. A graphical user interface was written using the Qt library. The program produced output in GNU LilyPond format, which was then processed by LilyPond to create scores and MIDI files.

## 4 Results

Qualitative experiments were performed by seeding the generator with a combination of random, uniform, and human-composed music. With a high number of generations, it became impossible to distinguish between different seeds. However, a low number of generations yielded music that was noticeably similar to the seed. Emphasizing various mutation parameters led to interesting results. For example, seeding the generator with J.S. Bach's first *English Suite* and emphasizing the re-meter operator gave a clever sounding rendition.

## 5 Future Work

Work on this project has some questions unanswered:

- To what extent does the input affect the time complexity of the algorithm?
- How do mutation rate and number of generations relate to each other? Which has the predominating effect (or are they both necessary)?
- Are genetic algorithms actually well-suited to this problem? Are they well-suited to any problem?

The project could be extended by building some sort of deeper musical knowledge into the generator. For instance, the generator has no notion of tonality, and does not create compositions that have any large-scale structure. Additionally, it would be ideal to have the program learn the correct rules of music by studying other music known to be good, rather than having the rules programmed in.

## 6 Conclusion

The question of whether or not computers can duplicate human creative work remains unanswered. Music produced by this program certainly did not sound as coherent as human-composed music, but it did not sound random, either. Greater understanding of the mental processes involved in human composition will certainly improve the quality of computer-generated music.

## References

- [1] *AI Methods for Algorithmic Composition: A Survey, a Critical View and Future Prospects*, Proceedings of the AISB'99 Symposium on Musical Creativity. AISB, 1999.
- [2] John H. Holland. *Hidden Order: How Adaptation Builds Complexity*. Perseus Books, 1995.
- [3] Peter M. Todd and Gregory M. Werner. Frankensteinian methods for evolutionary music composition. In *Musical networks: Parallel Distributed Perception and Performance*. MIT Press/Bradford Books, 1998.