# Automatic Generation of Multi-Instrument Musical Scores with Small and Large Scale Structure

## ABSTRACT

Many composers and researchers have attempted to program a computer to compose concert music scores. Frequently used methods include neural networks, petri nets, formal grammars, genetic algorithms, and Markov models. One limitation of technologies such as neural networks and Markov models is that they are entirely dependent on data, and can be too "overfitted" to produce an interesting variety of music. Another, arguably more important limitation of all of the above tools is that, while they produce music that is pleasing on a local level, they generally do not possess any structure – neither at the level of the phrase nor at the level of the piece as a whole.

To bypass these limitations, the author proposes a multi-paradigmatic, ensemble approach to automated music composition based on first principles. At this stage, in order to have an objective measure to judge the success of the project, the author is working with the goal of producing music in the 18th century classical style. Unlike in previous studies, the algorithms used are determined primarily by the guidelines of music theory and secondarily by the author's knowledge of available computational methods. Different tools are used to derive different aspects and levels of the composition, resulting in a composition that is both unique and coherent.

The smallest unit of structure in the program is the cell, an object 2 beats long that contains an array of pitches, an array of durations, a cell type (scalar, chordal or other) and (optionally) a chord. The chord must be previously specified. The durations are then determined from a probability distribution. Next, the cell type is decided, and then the pitches are determined based on the cell type and the chord.

While the first cells in a module are generated in the above fashion, most cells and larger structures are generated from "transformations" of previous material. Transformations are applied in a functional programming paradigm by treating different transforming functions as variables, and choosing stochastically among those variables. Examples of transformations include full and partial transposition, full and partial inversion, full and partial retrograde, and full and partial ornamentation or detraction. Transformations are applied after a chord progression for the new material is chosen. The program shuffles the list of variables representing the different transformations, and tries out different ones until it reaches a match based on a "fitness" check based on standard rules of voice leading as well as a check whether or not it fits the specified chord sequence.

The code is stored and generated hierarchically. The main data structure for storing and creating the music is a "Chunk," a treelike structure that contains additional fields for keeping musically relevant information. A Chunk may contain an array of sub-chunks (other, smaller Chunk elements), or may only contain an array of pitches, durations, and chord type (cells). Chunks function as smaller parts of phrases, and as larger modules. For instance, to generate a sonatina, the program generates first a Chunk corresponding to the exposition. The call to generate the exposition Chunk will in turn call the presentation chunk, which will call for a period, which will call for antecedents and consequents. All of the generating functions take as arguments previous material to be developed, so that the piece maintains a sense of unity.

Finally, after the monophonic Chunk corresponding to the entire piece is created, the piece is arranged according to the desired instrumentation. The arrangement of every sub-chunk is treated as a random variable, where larger sub-chunks are more likely to have varied arrangements than smaller sub-chunks. Predetermined accompaniment/arrangement types include Alberti eighths, sixteenths, and quarter notes; block chords, 5th species counterpoint, doubling the melody, and call and response.