MICA: A Hybrid Method for Corpus-Based Algorithmic Composition of Music Based on Genetic Algorithms, Zipf's Law, and Markov Models

by

Alan Nagelberg

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Engineering
The University of Manitoba
Winnipeg, Manitoba, Canada
January 2014

Thesis advisor Author

Dean K. McNeill **Alan Nagelberg**

MICA: A Hybrid Method for Corpus-Based Algorithmic Composition of Music Based on Genetic Algorithms, Zipf's Law, and Markov Models

Abstract

An algorithm known as the Musical Imitation and Creativity Algorithm (MICA) that composes stylistic music based on a corpus of works in a given style is presented. The corpus works are digital music scores created from the widely available MIDI format. The algorithm restricts the note placement in compositions using a Markov chain model built from discrete-time representations of the corpus pieces. New compositions are evolved using a genetic algorithm with a fitness function based on Zipf's Law properties of various musical metrics in the corpus pieces. The resulting compositions are evaluated by a panel of both musical and non-musical volunteers in a blind survey.

Contents

List of Figures

List of Tables

Acknowledgments

This work was made possible by the contributions of several people, to all of whom I am deeply indebted. I would like to begin by thanking my thesis advisor, Dr. Dean K. McNeill. Dr. McNeill showed a belief in my abilities early in my undergraduate studenthood, and he has helped secure funding for this research and prior projects. At the outset of my Master's degree, he allowed me to pursue this rather audacious and unconventional thesis project under his direction. Throughout the course of MICA's design and implementation, Dr. McNeill provided invaluable feedback and guidance to steer the course of the project in the right direction. I am truly grateful for all of my opportunities to work with Dr. McNeill, and for all of the Engineering knowledge and wisdom he has imparted.

Another person without whom this research would be impossible is Örjan Sandred. Mr. Sandred introduced me to the PWGL development environment, and to an entire world of computer music. He is a brilliant composer, with a wealth of knowledge about the ways in which computers can be used to assist in music composition. I extend my deepest thanks to Mr. Sandred for sharing his vast musical knowledge, and his constant willingness to help.

A group of the leading PWGL developers and composers gathers in Europe every year for a meeting known as Pedagogia e Ricerca Internazionale sui Sistemi Musicali Assistiti (PRISMA). I was fortunate enough to attend this meeting in the summer of 2012. At the meeting I was introduced to some brilliant people with similar research interests, I absorbed a great deal of information about development in PWGL, and learned about cutting edge computer music research happening around the globe. I would like to again thank Örjan Sandred for introducing me to this group, Jacopo

Baboni-Schilingi for organizing the PRISMA meeting, and all PRIMSA attendees for sharing their works.

My parents, being professional musicians, have been helping me understand music from as early an age as I can remember. They have always provided healthy encouragement to pursue my interests in mathematics and the sciences, and they have always supported whatever musical endeavours I pursue. Without their support and encouragement, this research would not be possible.

Recruiting volunteers to participate in a blind survey to evaluate MICA's output proved to be an elaborate task logistically, and it required the help of several people. Thank you to Sam Little, Örjan Sandred, Will Bonness, Dan Lockery, and Amy Dario for helping to make the blind survey a part of this project.

This thesis is dedicated to my parents, for their constant support and years of musical knowledge bestowed.

Chapter 1

Introduction

The goal of this research is to design and test an algorithm, referred to as the Musical Imitation and Creativity Algorithm (MICA), which takes as input a *training set* of digital music scores, and composes new pieces of music in the same style as those in the training set. The algorithm is intended to be a general-purpose computer composer that can be applied to a wide range of musical styles. Any music from the common-practice period (ie. the Baroque, Classical, and Romantic eras), and to some extent, more traditional Twentieth Century music should be within the scope of MICA's compositional capabilities. Furthermore, compositions produced by the program should exhibit some variation, or "creativity," across multiple runs. In other words, it should be able to compose multiple new compositions with significant variation for a given training set.

Algorithmic composition of music has a long and rich history, with one of the earliest examples being the oft-cited musical dice games of the eighteenth century (Hedges, 1978). These "games" involved rolling dice to randomly assemble a sequence of prewritten measures of music. The advent of electronic computers introduced a new palette of techniques to composers. The 1956 work *The Illiac Suite* (Hiller Jr and Isaacson, 1957) is generally considered to be the earliest musical composition composed by an electronic computer (McAlpine et al., 1999; Sandred et al., 2009). Hiller and Isaacson utilized various stochastic methods to compose the suite. Iannis Xenakis was another important pioneer of algorithmic composition, as he manually applied stochastic methods to compose music manually prior to the publication of the *The Illiac Suite*, but without the help of a computer (McAlpine et al., 1999).

As the accessibility of computational power grew quickly over the mid-to-late twentieth century, along with theoretical developments in Computer Science, more advanced algorithmic composition methods emerged. Cope's Experiments in Musical Intelligence project (Cope, 1992) based on generative grammars and motif-fixing is likely the first instance of a style-imitating algorithmic composition software. However, Cope's disclosure of his methods is somewhat incomplete (Collins, 2009, p. 109).

In recent years, Genetic Algorithms (GAs) have gained traction as a popular method for algorithmic composition, especially in interactive systems wherein the fitness function is formulated and altered by the user according to their personal preference, as in (Tokui and Iba, 2000; Moroni et al., 2000). Genetic Algorithms have seen less application in style-imitation systems, due to the fact that formulating a comprehensive fitness function algorithmically can be difficult (Nierhaus, 2009, pp. 182-184). The algorithm presented here overcomes the fitness function formulation problem by first employing a novel discrete-time representation of the training pieces to build a Markov chain model that can synthesize new pieces stochastically. The set of all possible sequences that may be produced by the Markov model then forms the search space for the GA. To find one of the "best" Markov sequences, the GA evolves a solution using a fitness function based on Zipf's Law properties (see Manaris et al., 2005) of the training data.

The algorithm is implemented in PWGL (Laurson et al., 2013), a visual music programming environment based on the Lispworks implementation of Common Lisp. The input training scores are passed to the software using PWGL's native Expressive Notation Package (ENP) file format (Laurson and Kuuskankare, 2003; Kuuskankare and Laurson, 2006), which can be produced directly from the widely available MIDI file format. The training data was obtained in the form of MIDI files from the public online databases *The Classical Archives* (`http://www.classicalarchives.com/`) and *The Mutopia Project* (`http://www.mutopiaproject.org/`).

Some simplifying assumptions are made about the nature of the input compositions. Every training piece must have a single time signature throughout the piece, with the exception of a pickup measure. Furthermore, the time signature of every training piece must be interchangeable by a scaling power of 2. For example, a training set containing pieces in $\frac{3}{8}$, $\frac{3}{4}$, and $\frac{3}{16}$ would be valid. Repeat signs are not handled by MICA, so repeats and first and second endings must be unfolded in the training pieces. The training pieces should each have the same number of voices. If all training pieces do not have the same number of voices, MICA finds the smallest number of voices in a training piece, and ignores any voices beyond that number in every piece. Only rhythm and pitch values and their metric position are examined by MICA. Compositional information beyond rhythm and pitch, such as dynamics, timbre, articulations, tempo changes, etc., are not incorporated into the algorithm. Any particular training set must be selected thoughtfully, so that there is sufficient commonality amongst the training data from which the algorithm can extract stylistic information.

The quality of MICA's compositions is inherently subjective. Therefore a group of volunteers with varying musical backgrounds participated in a blind survey that involved filling out a web form to evaluate characteristics of the program's output compositions, alongside historic compositions by authors of the training sets as control pieces. The survey results reveal that in terms of "musicality", MICA's compositions score similarly to comparable pieces by historic composers amongst musical and non-musical volunteers. Furthermore, volunteers with a musical background were often able to identify the composer whose compositions were used as training data for a given MICA composition.

The next chapter provides some of the theoretical background required to understand MICA's inner workings. Chapter 3 outlines the structural representation of musical scores within MICA, and Chapter 4 describes the algorithm itself in detail. Some example training sets are explored in Chapter 5. Results from the blind survey are examined in Chapter 6, Conclusions are given in Chapter 7, and possible future improvements to the algorithm are discussed in Chapter 8. Example MICA compositions are provided in the Appendix.

Chapter 2

Background

2.1 Markov Models in Music

Markov chains may be used in music to create a probabilistic model of the way in which sequences of musical objects move from one object to the next. Markov chains have been applied to music in several contexts (Jones, 1981; McAlpine et al., 1999).

A Markov chain is fully described by a *transition matrix*, P, and an *initial distribution*, $\boldsymbol{\pi_0}$. Implicit in the definition of a transition matrix and initial distribution is a *state space*, S, which is the finite alphabet of all values or "states" that are valid within the model. The transition matrix, P, is a *right stochastic matrix* that gives the probability of each state transitioning to any other state in a given process. The initial distribution is a probability mass function that gives the probability of each member of S being the initial state in a process.

This section demonstrates the typical use of Markov chains in algorithmic composition through examples. For a full treatment of Markov chain theory, see (Privault, 2013).

Example 2.1: Assume we wish to create a Markov chain model for the following sequence of notes:

$$C, D, E, C, D, E, C, D, E, C, D, D, C. \tag{2.1}$$

The state space for sequence 2.1 is $\{C, D, E\}$. Since there is only one input sequence, the initial distribution is

$$\boldsymbol{\pi_0} = [\pi_C, \pi_D, \pi_E] = [1, 0, 0] \tag{2.2}$$

where π_X is the probability of starting at note X. Counting the number of times each note in sequence 2.1 transitions to another yields the values in Table 2.1.

Table 2.1: Frequency of each note transition for example 2.1.

Current Note	**Next Note**	**Frequency**
C	D	4
D	E	3
E	C	3
D	D	1
D	C	1

We can use these frequencies to construct a transition matrix:

$$P = \begin{array}{c|ccc} & C & D & E \\ \hline C & 0 & 1 & 0 \\ D & 0.2 & 0.2 & 0.6 \\ E & 1 & 0 & 0 \end{array} \tag{2.3}$$

In the Markov chain described in Eq. 2.3, C always leads to a D, the probability of D leading to C is 0.2, the probability of D leading to D is 0.2, the probability of D leading to E is 0.6, and E always leads to C. This Markov chain must always begin in state C. Once a transition matrix is obtained, we can generate sequences of a given length by randomly selecting an initial state (which is always C in this case), and stochastically traversing the transition matrix to obtain the successive states. $\square$

The initial distribution in the previous example is trivial, as there is only 1 input sequence. Consider the next example, wherein there are multiple input sequences.

Example 2.2: Assume we wish to create a Markov chain model for the following three sequences of notes:

$$C, D, E, D, C. \tag{2.4}$$

$$E, C, D, E, D, D, C. \tag{2.5}$$

$$C, E, D, E, G. \tag{2.6}$$

The state space is now $\{C, D, E, G\}$, and the initial distribution is

$$\boldsymbol{\pi_0} = [\pi_C, \pi_D, \pi_E, \pi_G] = [\frac{2}{3}, 0, \frac{1}{3}, 0] \tag{2.7}$$

The state transition counts are given in Table 2.2.

Table 2.2: Frequency of each note transition for example 2.2.

Current Note	**Next Note**	**Frequency**
C	D	2
D	E	3
E	D	3
D	C	2
E	C	1
D	D	1
C	E	1
E	G	1

Normalizing the transitions out of each state yields the following matrix:

$$P = \begin{array}{c} \\ C \\ D \\ E \\ G \end{array} \begin{array}{c} \begin{array}{cccc} C & D & E & G \end{array} \\ \begin{bmatrix} 0 & 2/3 & 1/3 & 0 \\ 1/3 & 1/6 & 1/2 & 0 \\ 1/5 & 3/5 & 0 & 1/5 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{2.8}$$

Note that state G has zero transitions out of itself. Once state G is reached, there is no next state to which the process can proceed. In a valid transition matrix, every row must sum to 1. Therefore matrix 2.8 is an invalid transition matrix.

In the first example, the final note in the input sequence already contained a transition to some other note in the transition matrix. However in this example, state G appears for the first time at the end of a sequence, and never appears in the middle of a sequence.

If we wish generate sequences of arbitrary length using a transition matrix, there must be no states with zero transitions out, or else the output sequence will get "stuck" once it reaches such a state.

One heuristic solution to this problem is to wraparound the last element of the input sequence to the first element in the sequence. Alternatively, one could pick a random state in the existing state-space as the next state. Such methods are only necessary when the last element in a sequence does not yet exist in the state space. The method MICA uses to handle this scenario is outlined in Section 3.2.1. □

Markov chains are fundamental to MICA's operation. However, an understanding of MICA's method for encoding a piece of music is necessary before the role of Markov chains can be discussed. This encoding and its usage with Markov chains is discussed in Section 3.2.

The next section provides background on Zipf's Law and its applications to music.

2.2 Zipf's Law in Music

George Kingsley Zipf (1949) examined several natural language texts, such as Homer's *Illiad* and James Joyce's *Ulysses*, and found a relationship between the *frequency* of occurrence of a given word in a body of text, and the *rank* of that word. The rank of a word in a group of words is defined to be the position of the word in a list of all words in the group ordered by frequency, with rank 1 being the most frequent, rank 2 being the second most frequent, etc. Zipf's relationship is

$$F \sim r^{-s} \tag{2.9}$$

where F is the frequency of a word, r is the rank of that word, and s is an exponent characterizing the distribution. Zipf describes the case where $s = 1$ as the "ideal" case. Restated as a probability mass function (PMF), Zipf's law is

$$P(r; s, N) = \frac{1}{r^s H_{N,s}} \tag{2.10}$$

where r is the rank, s is the exponent characterizing the distribution, N is the total number of ranked events, $P(r; s, N)$ is the probability of an event of rank r for a given s and N, and $H_{N,s}$ is defined by

$$H_{N,s} = \sum_{n=1}^{N} \frac{1}{n^s} = 1 + \frac{1}{2^s} + \frac{1}{3^s} + \cdots + \frac{1}{N^s}. \tag{2.11}$$

The quantity $H_{N,s}$ is known as the generalized harmonic number of order N of s, and it is present in Eq. 2.10 to normalize the PMF, ensuring that the values of the PMF sum to 1 across all r for a given set of ranked events. When this rank-frequency relationship is plotted on a log-log plot, the result is a straight line with slope $-s$.

Zipf goes on to observe this relationship occurring in many natural and social phenomena, including music (1949). He investigates the rank-frequency distribution of the melodic intervals as well as the "number of intervals of like length between the repetition of notes" in Mozart's Bassoon Concert in B♭ Major.

Inspired by Zipf's efforts, Manaris et al. (2005) applied Zipf's Law to a wide variety of musical metrics to perform algorithmic author attribution, style identification, and "pleasantness" prediction of musical inputs. The Zipf's Law characteristics of each musical metric were used as training data for an artificial neural network (ANN).

After training, the ANN was able to attribute the authors of compositions, identify style, and predict "pleasantness" of new input with high accuracy.

The method used by Manaris et al. involved plotting the rank-frequency distribution of each metric on a log-log plot, and performing a simple linear regression to find the line of best fit. To characterize a given rank-frequency distribution, Manaris et al. used the slope of this line, m, and the coefficient of determination, R^2. The R^2 value is the square of the correlation coefficient, and is therefore bounded by $0 \leq R^2 \leq 1$. The R^2 value indicates how well the data fits a straight line, with 0 being the worst fit, and 1 being a perfect fit.

The value of m along with R^2 provide some information about the "Zipfian" characteristics of a given metric. The m and R^2 values for every musical metric served as the inputs to the ANNs used by Manaris et al. In MICA, these same values are used to characterize each musical metric, and they are central to the fitness function used by MICA's genetic algorithm.

2.3 Genetic Algorithms

Genetic Algorithms (GAs) mimic the process of biological evolution in an attempt to find the global maximum or minimum of some complicated function. In general, the process of maximizing or minimizing a function is known as *mathematical optimization*, and the function being optimized is known as the *objective function*, or in GA terminology, the *fitness function*.

A GA is not guaranteed to find the absolute global maximum or minimum, but rather it settles for finding an approximate solution in an efficient manner. This type of algorithm is referred to as a *heuristic algorithm*. Heuristic algorithms for optimization such as GAs are useful when the maxima or minima of the fitness function cannot be found analytically, and it is impractical to generate every possible solution to determine the global maximum or minimum.

A flow-chart for a typical GA is shown in Fig. 2.1 The algorithm begins by first randomly generating a group of candidate solutions in the GA's search space. The candidate solutions are called *chromosomes*, and the group of candidate solutions is

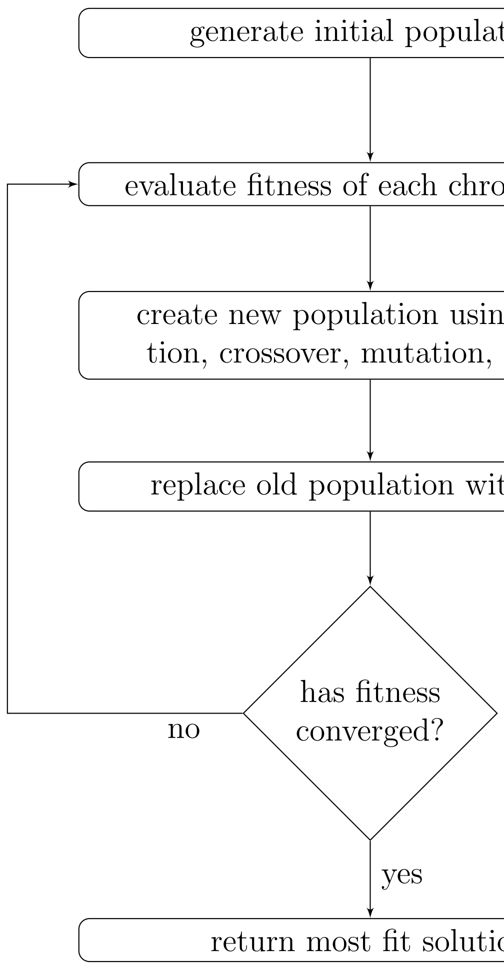

generate initial population
evaluate fitness of each chromosome
create new population using selection, crossover, mutation, elitism
replace old population with new
has fitness converged?
no
yes
return most fit solution

Figure 2.1: Flow chart of a typical genetic algorithm, adapted from (Kumar and Jyotishree, 2012).

called the *mating pool* or the *population*. Naturally, GA nomenclature is inspired by the biological analogy. After generating the initial mating pool, the fitness of each chromosome is evaluated by calling the fitness function on each chromosome and storing the result. A series of *genetic operations* are then applied to the mating pool to create the next *generation* of chromosomes. These operations are applied to each new population repeatedly, improving the overall fitness with each iteration, until the some termination condition is met.

The first of the genetic operations is *selection*. Selection is the process by which chromosomes are propagated into the next generation. Various selection schemes exist (Goldberg and Deb, 1991; Miller and Goldberg, 1996), but the scheme known as *roulette wheel selection*, or *proportionate selection* is probably the most well-known (Thierens and Goldberg, 1994), and is the scheme used by MICA. In roulette wheel selection, the probability of a chromosome being selected is proportional to the chromosomes fitness. Therefore the probability of chromosome i being selected is

$$P_i(t) = \frac{f_i(t)}{f_{total}(t)} \tag{2.12}$$

where $P_i(t)$ is the probability of selecting chromosome i at time t, f_i is the fitness of chromosome i at time t, and $f_{total}(t)$ is the sum of the fitness of every chromosome in the mating pool at time t. This form of selection favours the most fit solutions, but allows for the possibility of some latent features of less fit solutions to be propagated into the next generation.

After a pair of chromosomes is selected, the chromosomes probabilistically undergo a *crossover* operation. The crossover operation combines the two chromosomes to form two new children. The simplest form of crossover is the single-point crossover. In this case, a single common position in the pair of chromosomes is randomly selected, and the two chromosomes are split at this position and recombined with each other to form two new chromosomes. Another way to combine two chromosomes is a two-point crossover. In this case, two positions are randomly selected within the chromosomes, and each chromosome is split at the crossover points and combined with a section from the other chromosome to form two new chromosomes.

Other methods for crossing over two chromosomes exist, most of which are dependant on the nature of the search space. The two aforementioned crossover techniques are the two used by MICA. The crossover operation occurs with some fixed crossover probability, p_c.

The next genetic operation is *mutation*. This is the random alteration of each chromosome. The GA has a fixed mutation probability, p_m, which is the probability of a given chromosome being mutated.

The precise values of p_c and p_m are dependent on the specifics of the problem the GA is solving. Methods using adaptive crossover and mutation probabilities exist, which vary p_c and p_m with the relative fitness of the chromosomes (Srinivas and Patnaik, 1994). However, MICA does not currently employ such methods. The numerical crossover and mutation probabilities used by MICA are given in Section 4.4.2 and 4.4.3.

The last genetic operation is *elitism*, whereby the two most fit solutions in a given generation are always copied into the next generation unchanged. This ensures that the two most fit solutions are never lost from one generation to the next.

The GA may be terminated in one of several ways: (1) when a certain fitness value is reached; (2) after a specific number of generations; (3) after a specified duration; (4) when the best fitness value reaches a plateau, meaning a certain number of consecutive generations occur with no fitness improvement. Option (4) is the most scalable and reliable option across different fitness functions, and it is the termination method employed by MICA.

Chapter 3

Musical Score Representation

The effectiveness of any algorithmic composition system is largely dependent upon the internal data structures used to represent a musical score. Two models of scores are used by MICA.

3.1 Standard Score Representation

The Unified Modeling Language (UML) diagram in Fig. 3.1 depicts the standard musical score representation within MICA. A *Piece* object consists of an array of *Voice* objects, each Voice object consists of an array of *Measure* objects, and each Measure object consists of a linked-list of *Note* objects, and each Note object has *Pitch* and *Rhythm* properties. The Rhythm class has properties to indicate whether the note is a rest, a tied note, or a grace note. The Pitch class has zero or more integer pitches that represent the MIDI note number of the note. If the number of pitches is zero, this implies that the note is a rest.

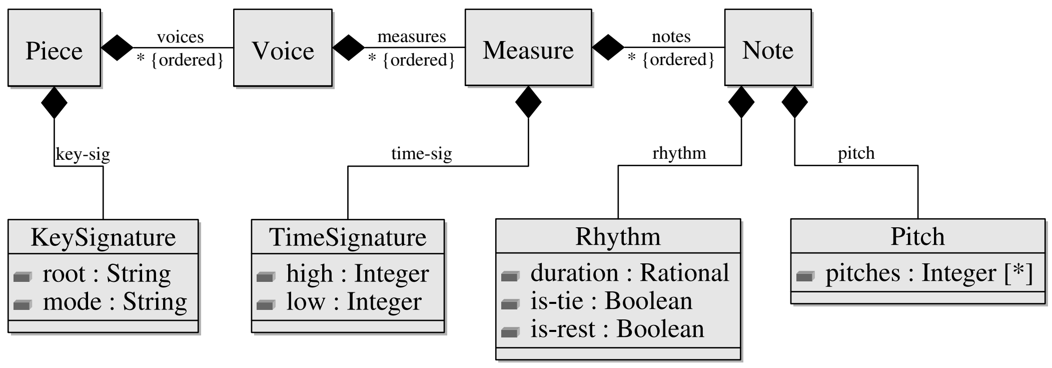


Figure 3.1: Simplified UML diagram of the standard musical score representation.

The collection of Note objects within each Measure are stored in a linked-list because throughout the duration of the program, the length of this list may change, and notes may be added or removed[1].

The structure shown in Fig. 3.1 arises from applying standard object-oriented design principles to develop an abstraction of a musical score. Therefore it is the representation one would expect in typical musical score software. The discrete-time representation presented in the next chapter is less similar to a human interpretation of a score. In exchange, the discrete-time representation offers attractive features for computer analysis.

3.2 Discrete-Time Score Representation

The discrete-time representation of a musical score is illustrated in Fig. 3.2. Each measure of the score is divided into a fixed number of subdivisions. For a given training set in MICA, this subdivision is fixed. The subdivision is calculated by

[1]This is mainly because of a legacy mutation function. An array of notes is just as effective as a linked list under MICA's current mutation function.

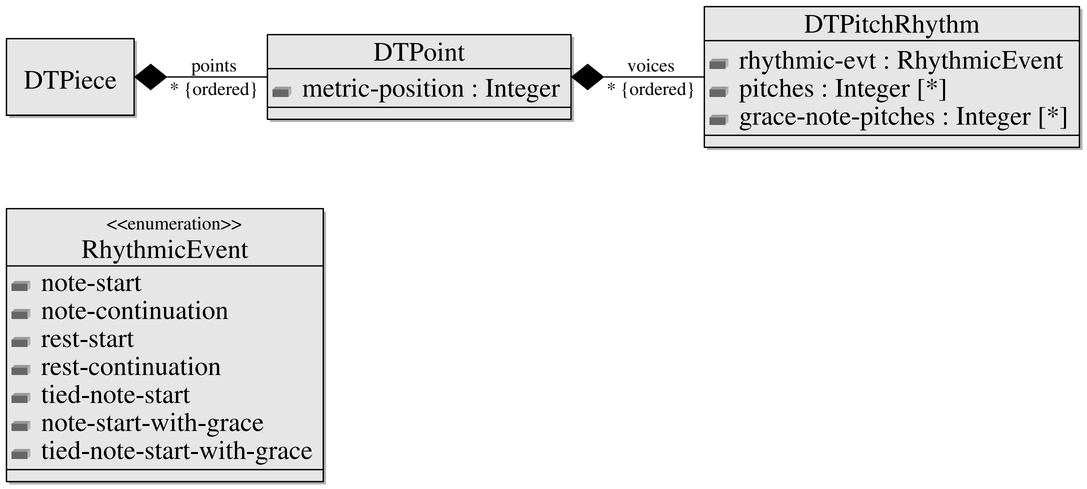


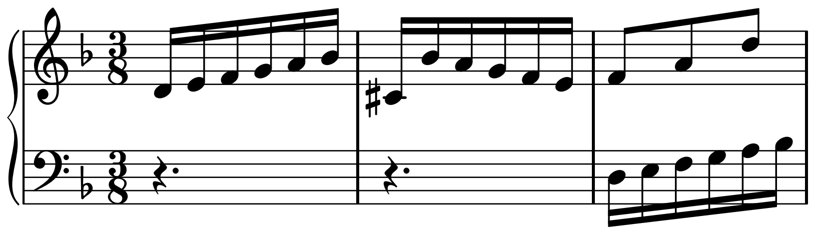
Figure 3.2: Discrete-time musical score representation UML Diagram.

finding the least common multiple of the denominator of every rhythmic duration in the training set. In set theory notation,

$$subdivision = lcm(\{d : d \in \{denom(r) : r \in R\}\}) \tag{3.1}$$

where $lcm(x)$ is the least common multiple of integer x, R is the set of fractional rhythmic durations in every training piece, and $denom(y)$ is the denominator of rational y.

Fig. 3.3 illustrates how a score translates into the discrete-time representation. Each voice, consisting of a pair of *rhythmic-event* and *pitch* values, corresponds to a DTPitchRhythm object from the UML diagram in Fig. 3.2. Since there are no grace notes in this example, the *grace-note-pitches* property is omitted in Fig. 3.3.



subdivision=16

$\Downarrow$

		0	1	2	3	4	5	...
	metric-potision:	0	1	2	3	4	5	...
voice0	**rhythmic-event:**	N.S.	N.S.	N.S.	N.S.	N.S.	N.S.	...
	pitch:	(62)	(64)	(65)	(67)	(69)	(70)	...
voice1	**rhythmic-event:**	R.S.	R.C.	R.C.	R.C.	R.C.	R.C.	...
	pitch:	()	()	()	()	()	()	...

0	1	2	3	4	5	0	1	2	3	4	5
N.S.	N.S.	N.S.	N.S.	N.S.	N.S.	N.S.	N.C.	N.S.	N.C.	N.S.	N.C.
(61)	(70)	(69)	(67)	(65)	(64)	(65)	(65)	(69)	(69)	(74)	(74)
R.S.	R.C.	R.C.	R.C.	R.C.	R.C.	N.S.	N.S.	N.S.	N.S.	N.S.	N.S.
()	()	()	()	()	()	(50)	(52)	(53)	(55)	(57)	(58)

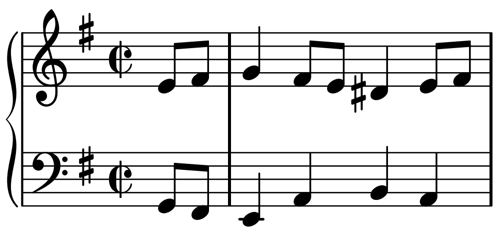
Figure 3.3: An example of how the discrete-time representation works, using the first 3 measures Bach's Two-Part Invention No. 4 (BWV 775). Score fragment adapted from http://www.mutopiaproject.org/.

The rhythmic-event abbreviations used in Fig. 3.3 are:

- N.S. - Note start
- N.C. - Note continuation
- R.S. - Rest start
- R.C. - Rest continuation

The encapsulation of voice0, voice1, and a metric position corresponds to a *DT-Point* object. Each DTPoint object is surrounded by a solid rectangle in Fig. 3.3. The contents of a DTPoint is said to occupy a *time slice*. A collection of DTPoint objects comprises a complete DTPiece, which represents an entire score. This representation can be extended to any number of voices.

Time signature changes within a piece are not permitted in MICA, but pick-up measures are supported. A pick-up measure is a measure at the beginning of the piece that is shorter than the specified time signature. Metric position within pick-up measures in MICA is defined such that the pick-up measure ends at the same metric position as it would with a regular full-length measure in the given time signature, as illustrated in Fig. 3.4.



subdivision=8

$\Downarrow$

	metric-potision:	6	7	0	1	2	3	…
voice0	**rhythmic-event:**	N.S.	N.S.	N.S.	N.C.	N.S.	N.S.	…
	pitch:	(64)	(66)	(67)	(67)	(66)	(64)	…
voice1	**rhythmic-event:**	R.S.	R.C.	R.C.	R.C.	R.C.	R.C.	…
	pitch:	(55)	(54)	(52)	(52)	(57)	(57)	…

Figure 3.4: An example of a pickup measure in the discrete-time representation, using Bach's Bourrée in E Minor (BWV 996). Score fragment adapted from http://www.mutopiaproject.org/.

Note that MICA stores both the standard representation and the discrete-time representation of each piece internally. The software ensures that any changes made to one representation are updated in the other representation at an appropriate time.

3.2.1 Markov Chain Model

The discrete-time representation of musical scores forms the basis of a Markov Chain model of the training set data. This Markov Chain model determines which musical events are valid in terms of pitch, rhythm, and metric position.

The Markov Chain model is built by first creating the discrete-time representation of all of the pieces in the training set. Then these discrete-time representations are used to build a Markov Chain transition matrix which will be traversed stochastically to generate new sequences.

Every DTPoint object in every training piece object is considered to be a state in a Markov process, and each training piece is considered to be a sequence that was generated by stochastically traversing the same first-order Markov transition matrix. Using these assumptions, the transition matrix and corresponding initial distribution can be built from the training pieces in their discrete-time representations.

The last measure of each piece is given special consideration. The Markov Chain analysis ends at the last DTPoint in the second last measure of each training piece, because the final measure is copied directly from one of the training pieces. If this last DTPoint is not already present in the transition matrix, MICA will wraparound the sequence to the first DTPoint with a metric position of 0, to overcome the problem of getting "stuck" as described in Section 2.1. The first DTPoint with metric position of 0 is used to ensure a consistent metric structure, because there may be a pickup measure at the beginning of the piece. As described earlier in this section, pickup measures start at a non-zero metric position.

The last measure from each piece is stored in an array. While synthesizing a new piece, one of the last measures is randomly selected and appended to the end of a Markov Chain sequence.

Since some training pieces may have pickups of different lengths, and others no pickup measures at all, the initial state is dependent on which skeleton piece is used for the composition. In each MICA composition, a skeleton piece from the training set is selected, and its metric structure used in the new composition. The initial DTPoint may be the DT point from any piece in the training set that begins at the same metric position as the skeleton piece.

The transition matrix of DTPoints is very sparse, with most states having only one or two possible next states. Therefore, every row is implemented as a linked list, with zero-probability entries not present. The rows are initially created as a linked list of rows, as the total number of rows is not known at the outset. After every row is created, the list of rows is coerced to an array, and sorted by a simple order on the DTPoint associated with each row. This is so a binary search can be used to access any row rather than a linear search, since rows will be accessed many times throughout the course of the algorithm. Another viable implementation for the list of rows would be a hash table.

Chapter 4

The Algorithm

A high-level flow chart of MICA is presented in Fig. 4.1. The purpose of the sections in this chapter is to explain in detail each step of this algorithm.

4.1 Preprocessing the Training Pieces

Before MICA can extract meaningful information from the training pieces, they must be transposed to a common key or mode, and transposed metrically so that the significance of metric position may be captured.

4.1.1 Key Transposition

Transposing all pieces to the same key, for instance C-major, presents a problem. Suppose one training piece is in G-major, and another in F-major, and they both contain pitches in a similar range. It would make the most sense to either transpose the F-major piece to G-major, or the G-major piece to F-major, or even both pieces

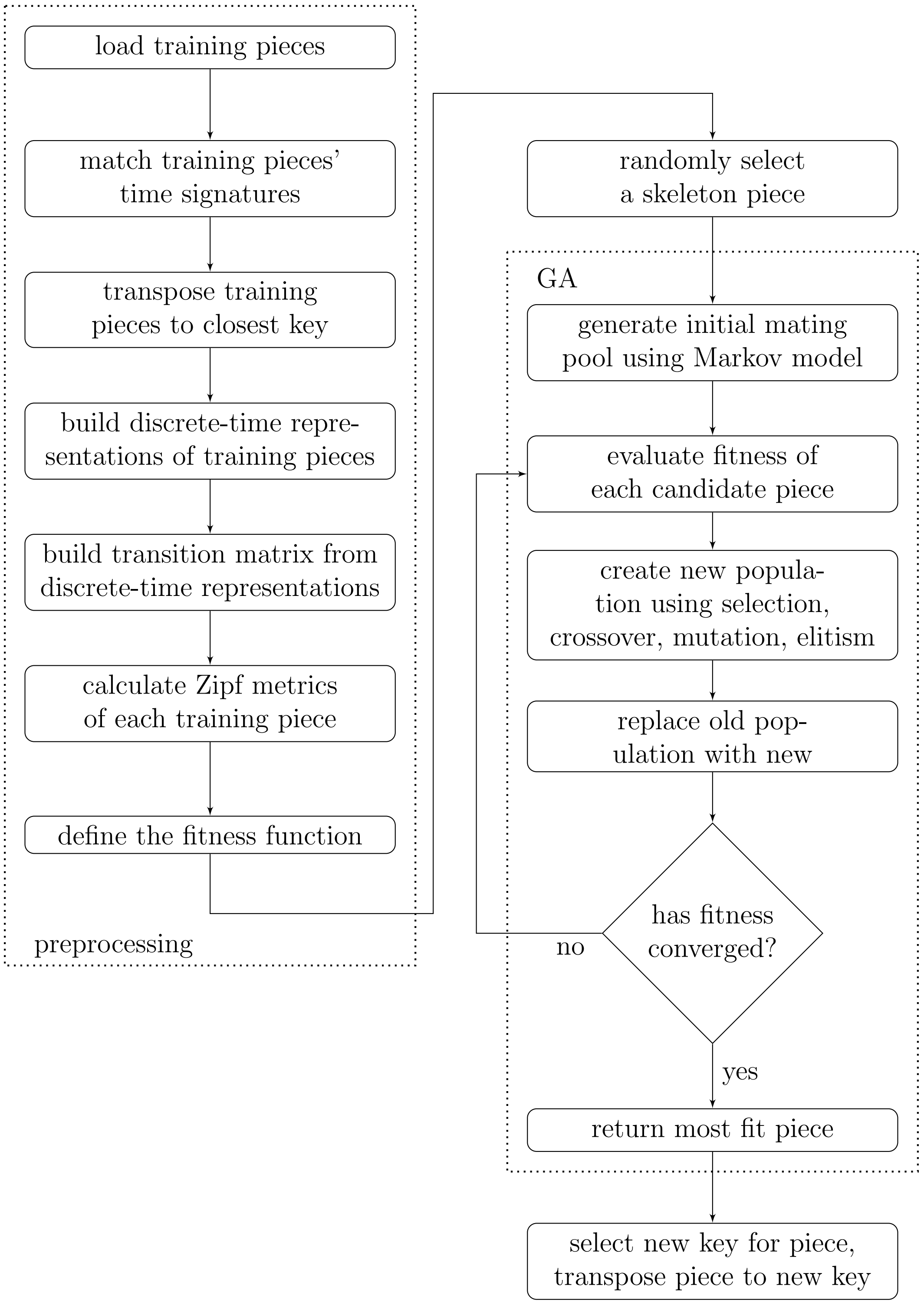

load training pieces
match training pieces' time signatures
transpose training pieces to closest key
build discrete-time representations of training pieces
build transition matrix from discrete-time representations
calculate Zipf metrics of each training piece
define the fitness function
preprocessing
randomly select a skeleton piece
GA
generate initial mating pool using Markov model
evaluate fitness of each candidate piece
create new population using selection, crossover, mutation, elitism
replace old population with new
has fitness converged?
no
yes
return most fit piece
select new key for piece, transpose piece to new key

Figure 4.1: High-level flow chart of MICA.

to F♯-major. Transposing both pieces to C-major would result in moving each of them a perfect fourth in the opposite direction, which could make a large portion of the musical content in the two pieces different by 1 octave.

To understand how MICA decides the key to which it should transpose the training pieces, we must define the *key tranposition distance*.

Definition: We define the *key tranposition distance* $D_K(X, Y)$ from key X to key Y be the smallest number of half steps required to transpose key X to Y. Here are a few examples of $D_K(X, Y)$ for different values X and Y:

$$D_K(\text{C-major}, \text{D-major}) = 2 \tag{4.1}$$

$$D_K(\text{A-major}, \text{E-major}) = -4 \tag{4.2}$$

$$D_K(\text{B-minor}, \text{D-minor}) = 3 \tag{4.3}$$

In the case of a tritone transposition, the value of $D_K(X, Y)$ may be plus or minus 6. An arbitrary but consistent choice can be made to use either the positive or negative value.

When the two keys X and Y are not in the same mode, ie. one is major and the other is minor, the relative major (or minor) of Y is used to determine $D_K(X, Y)$. However, when using training pieces in different modes, MICA will likely produce musically confused and incoherent results. For example, using both major and minor Bach Two-Part Inventions together in a training set yielded unconvincing output compositions.

In (Schwanauer and Levitt, 1993), Cope suggests transposing all pieces to the same mode (p. 406), but does not propose an algorithmic method for doing so. In general, transposing a minor piece to its enharmonic major is a non-trivial task, since accidentals need special treatment depending upon their musical context. For instance, a brief and temporary key modulation in a piece could complicate this matter significantly.

In a separate publication, Cope (1992) presents a method for coercing short motifs that he calls "signatures" into a diatonic key (either major or minor) by randomly altering any out of key notes by ± 1 semi-tone, with an equal chance of either value (p. 78). In the context of MICA, this method would distort and perhaps ruin the original musical ideas by obliterating any accidentals. A general purpose or universal method for changing a major piece to minor, and vice versa, is a topic of interest, and requires further research.

4.1.2 Metric Transposition

The time signature of every piece in a given training set is assumed to be closely related. Specifically, it is assumed that every time signature is interchangeable after scaling the rhythmic durations by some power of 2. Under this assumption, metric transposition changes the time signature of every piece in the training set to match the shortest time signature present.

An alternative method to scaling is the addition of bar lines without scaling the rhythmic durations. For example, $\frac{4}{4}$ could be made into $\frac{2}{4}$ by adding a bar line in the middle of each $\frac{4}{4}$ measure. This method could be used in tandem with scaling as well.

Deciding which method is best for a given training set is a difficult task. It requires an inspection of the distribution of rhythmic durations in each piece, as well as the time signatures present. Currently, MICA does not attempt this form of "intelligent" metric transposition, but rather uses the scaling method exclusively. Therefore, it is recommended that the time signature of every training has at least the same number of beats per measure, while allowing different beat values. For example, the training set may contain pieces in both $\frac{3}{8}$ and $\frac{3}{4}$.

4.2 Mating Pool Generation

A candidate piece is generated by traversing the Markov chain created from the discrete-time representation described in Section 3.2.1. For each execution of the algorithm, a skeleton piece is randomly selected from the training set. The metric structure of this piece is used for every candidate piece. This skeleton piece determines the necessary length of the sequence to be generated from the Markov transition matrix, as well as the metric position of the initial state. When there is more than one piece in the training set that begins at the same metric position as the skeleton piece, the initial DTPoint is selected proportionally from every such piece.

The sequence of DTPoints is generated up to the second last measure, then a final measure is appended to the piece. The final measure is randomly selected from one of the training pieces uniformly. The selection of more appropriate final measures is handled by the fitness function, as described in Section 4.3.3.

The set of all possible sequences generated using the transition matrix, along with a final measure, forms the search space for MICA's genetic algorithm.

4.3 Fitness Function

There are four major components to the fitness function used by MICA's GA: (1) global Zipf metrics; (2) subdivided Zipf metrics; (3) the check for validity of the transition into the final measure; and (4) prevention of MICA's output matching a training piece exactly. The fitness of a candidate piece C is

$$f(C) = -w_g d_g(C,T)_{min} - w_s d_s(C,T)_{min} + w_l l(C) - w_e e(C,\mathbf{T}) \tag{4.4}$$

where w_g is the weight of the global Zipf metrics, $d_g(C,T)_{min}$ is the minimum weighted Euclidean distance from C to a training piece T for the global Zipf metrics, w_s is the weight of the subdivided Zipf metrics $d_g(C,T)_{min}$ is the minimum weighted Euclidean distance from C to a training piece T for the subdivided Zipf metrics, w_e is the punishment weight for an exact match, $e(C,\mathbf{T})$ is the exact match test, w_l is the weight of last measure validity, and $l(C)$ is the last measure validity test.

The remainder of this section describes the four components of the fitness function.

4.3.1 Global Zipf Metrics

The fitness function primarily is based on the Zipf's Law properties of a group of musical metrics. These musical metrics are listed in Table 4.1. In addition to applying the metrics across an entire piece, as with global Zipf metrics, MICA applies some metrics recursively to subdivisions of the piece, down to a granularity of 1 measure.

The weighted Euclidean distance from a candidate piece C to a training piece T

Table 4.1: Complete list of global Zipf metrics used in MICA. Most metrics adapted from (Manaris et al., 2005, p. 58).

Metric	Description
Pitch	Rank-frequency distribution of the 128 MIDI pitches
Pitch class	Rank-frequency distribution of the 12 pitch classes
Pitch Distance	Rank-frequency distribution of length of time intervals between pitch repetitions
Pitch Class Distance	Rank-frequency distribution of length of time intervals between pitch-class repetitions
Pitch Duration	Rank-frequency distribution of pitch durations
Pitch Class Duration	Rank-frequency distribution of pitch class durations
Rhythmic Duration	Rank-frequency distribution of rhythmic durations of both pitches and rests
Melodic Interval	Rank-frequency distribution of melodic intervals within each voice
Melodic Interval Order-2	Rank-frequency distribution of difference of melodic intervals
Melodic Interval Order-3	Rank-frequency distribution of difference of melodic intervals order-2
Melodic Interval Order-4	Rank-frequency distribution of difference of melodic intervals order-3
Melodic Interval Order-5	Rank-frequency distribution of difference of melodic intervals order-4
Melodic Interval Order-6	Rank-frequency distribution of difference of melodic intervals order-5
Harmonic Interval	Rank-frequency distribution of harmonic intervals within each chord, from the lowest note
Harmonic-Melodic Interval	Rank-frequency distribution of harmonic and melodic intervals combined
Harmonic Consonance	Rank-frequency distribution of harmonic intervals within chord based on music theoretic consonance
Pitch Class Harmony	Rank-frequency distribution of pitch classes played simultaneously

for the global Zipf metrics is

$$d_g(C,T) = \sqrt{\sum_{i=1}^{N} \{w_{m,i}[m_i(C) - m_i(T)]^2 + w_{R^2,i}[R_i^2(C) - R_i^2(T)]^2\}} \tag{4.5}$$

where N is the total number of metrics, $m_i(C)$ is the slope of the i^{th} metric of the candidate piece, $R_i^2(C)$ is the R^2 value of the i^{th} metric of the candidate piece, $m_i(T)$ is the slope of the i^{th} metric of the training piece, $R_i^2(T)$ is the R^2 value of the i^{th} metric of the training piece, $w_{m,i}$ is the weight of the slope for the i^{th} metric, and $w_{R^2,i}$ is the weight of the R^2 value for the i^{th} metric.

There is no "correct" value for the weight parameters, they are specified and adjusted experimentally. Manaris et al. (2005) specify the largest 13 ANN weights in their "pleasantness" prediction experiment (p. 67). These weights are used as a starting point for the weights in Eq. 4.5.

To obtain a numerical fitness contribution of a candidate piece, MICA uses the closest training piece's weighted Euclidean distance, as in Eq. 4.5. This implies that for a given candidate piece, MICA must find the weighted Euclidean distance from the candidate piece to every training piece, select the smallest distance, and multiply it by -1 since the GA is maximizing the fitness function, not minimizing. This value is then multiplied by the relative weight of the global metrics, w_g, and added to the total fitness.

4.3.2 Subdivided Zipf Metrics

In addition to applying Zipf's Law metrics across an entire piece, as is the case with global Zipf metrics, MICA applies some metrics recursively to subdivisions of the piece, down to a granularity of 1 measure. These metrics are referred to as *subdivided Zipf metrics*.

Fig. 4.2 illustrates the way a piece is recursively subdivided. The recursive subdividing ends when the subdivision on the far left has a length of one measure. The shortest piece in the training set is used to determine the depth of the subdivisions, to ensure that a piece is never subdivided into sections smaller than one measure. The powers of 2 between which the length of the shortest piece lies determines the number of subdivisions. For example, if the length of the piece is between 8 and 15 inclusively, the maximum depth of subdivisions, D, is 2. If the length of the piece is between 16 and 31 inclusively, the maximum depth is $D = 3$, etc. Once the maximum depth is determined, it remains the same across all training pieces and candidate pieces.

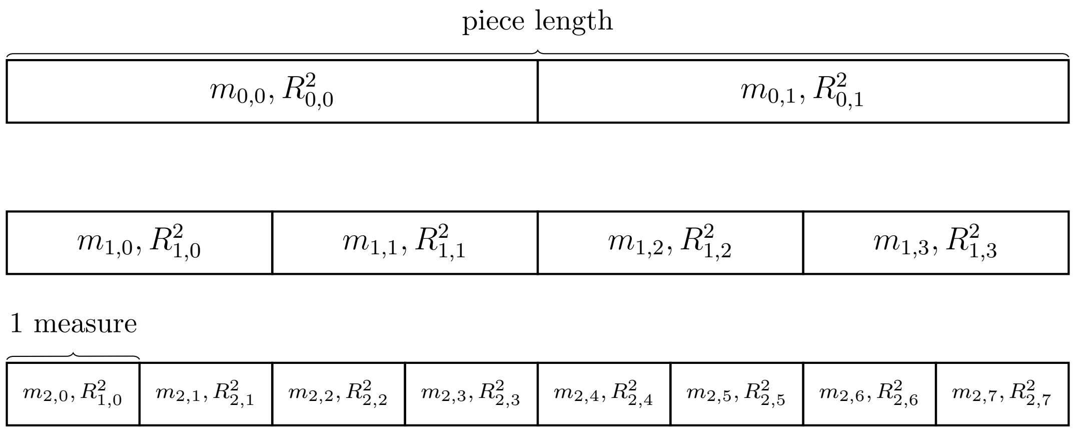


Figure 4.2: The subdivided Zipf metrics. The recursive subdividing ends once the width of the far left subdivision is 1 measure.

The subdivided sections are referenced by their starting index (a measure number), and their length in measures. This pair of values is encapsulated and referred to as a *start-index-length-pair* object. At any given depth of subdivisions, d, the number of start-index-length-pairs is 2^d. These start-index-length-pairs indicate the measures over which the Zipf metrics should be evaluated.

The algorithm for finding the subdivision points at a given depth is shown in Fig. 4.3. The input parameters to the algorithm in Fig. 4.3 are:

- d - the depth of subdivision.
- a - the array into which the start-index-length-pairs are stored. The total size of this array must be a power of 2.
- i - the current starting index of a start-index-length-pair.
- len - the current length of a start-index-length-pair.
- j - the index in a into which the start-index-length-pair should be stored.
- $aLen$ - the length of the current sub-array of a.

Since the length of every piece is not necessarily a power of 2, most pieces will not subdivide perfectly down to 1 measure subdivisions. Some subdivisions will have different lengths at any given depth. The varying length of the subdivisions is governed by the recursive algorithm in Fig. 4.3.

The function in Fig. 4.4 simply stores the points calculated by the function in Fig. 4.3. The input parameters to the algorithm in Fig. 4.4 are: d, the current depth of subdivision and len, the length of the piece in measures.

```
function GET-START-LENGTH-PAIRS($d$, $a$, $i$, $len$, $j$, $aLen$)
    if $d \geq 0$ then
        GET-START-LENGTH-PAIRS($d-1$, $a$, $i$, $\lfloor \frac{len}{2} \rfloor$, $j$, $\frac{aLen}{2}$)
        GET-START-LENGTH-PAIRS($d-1$, $a$, $i + \lfloor \frac{len}{2} \rfloor$, $\lceil \frac{len}{2} \rceil$, $j + \frac{aLen}{2}$, $\frac{aLen}{2}$)
    else
        $a[j] \leftarrow$ MAKE-START-INDEX-LENGTH-PAIR($i$, $len$)
    end if
end function
```

Figure 4.3: Pseudocode for recursively determining the subdivision measure start indices and their respective lengths at a given depth of subdivision.

```
function CREATE-START-LEN-PAIRS-ARRAY($d$, $len$)
    $a \leftarrow$ make array of length $2^{d+1}$
    GET-START-LENGTH-PAIRS($d$, $a$, 0, $len$, 0, length($a$))
    return $a$
end function
```

Figure 4.4: Pseudocode for determining the subdivision measure start indices and their respective lengths. This code makes a call to the function in Fig. 4.3.

For each piece, create-start-len-pairs-array is called at each depth of subdivision. As mentioned earlier, the maximum depth is determined by the length of the shortest piece in the training set. The start-index-length pairs found by the algorithm for a given piece may be reused by any piece of the same length, such as the candidate pieces created from a selected skeleton piece. The algorithm uses these start-index-length pairs to obtain a subsequence of the piece (a sequence of measures), then it evaluates the Zipf metric over each subsequence and stores the m and R^2 values for each subdivision.

Similar to the global Zipf metrics, the subdivided Zipf metrics use a weighted Euclidean distance for fitness calculation, but the weight parameter of any m or R^2 value is predetermined by the depth at which the subdivision lies. The weight of each metric at any given depth is the reciprocal of the total number of subdivisions at that depth. For example, when the piece is divided into 2 subdivisions, the weight of each subdivision metric is 1/2. When divided into 4 parts, the weight of each subdivision metric is 1/4, etc. This gives each level an equal weight across the sum of all subdivisions at a given depth.

The equation for the weighted Euclidean distance from a candidate piece C to a training piece T for a single subdivided Zipf metric is

$$d_s(C,T) = \sqrt{\sum_{i=0}^{D} \left\{ \frac{1}{2^{i+1}} \left\{ \sum_{j=0}^{2^{i+1}-1} [m_{i,j}(C) - m_{i,j}(T)]^2 + [R^2_{i,j}(C) - R^2_{i,j}(T)]^2 \right\} \right\}} \quad (4.6)$$

where D is the depth of subdivision, $m_{i,j}(C)$ and $R^2_{i,j}(C)$ are the slope and R^2 values of the jth metric at depth i of a candidate piece, respectively (see Fig. 4.2), and $m_{i,j}(T)$ and $R^2_{i,j}(T)$ are the slope and R^2 values of the jth metric at depth i of a training piece, respectively. The weight of every metric at depth i is $2^{-(i+1)}$, since depth 0 corresponds to the piece being subdivided into 2 parts.

4.3.3 Final Measure Validity

Recall from Section 3.2.1 that the final measure of any candidate piece is simply appended to the end of a generated Markov sequence. At the time of the piece's initial generation, no check is performed as to the validity of the last measure. Therefore, to ensure the piece enters the final measure in accordance with the musical rules of the training set, a small fitness bonus is awarded when either:

1. The last DTPoint in the second last measure and the first DTPoint of the last

measure are the same as a pair from the training set, or;

2. The last DTPoint in the second last measure and the first DTPoint of the last measure have a valid transition in the transition matrix.

4.3.4 Exact Match Prevention

Since MICA seeks to compose new musical material that is stylistically representative of the training set, we do not want MICA to simply reproduce one of the training pieces, as this would not really be a "composition." Therefore, in the event that MICA regenerates one of the training pieces precisely, a large punishment weight, w_e, is subtracted from that pieces fitness value, and it is effectively removed from the mating pool.

An exact match is determined by checking the weighted Euclidean distance of the subdivided Zipf metrics. If the subdivided Zipf metrics Euclidean distance from Eq. 4.6 from a candidate piece to training piece is smaller than some very small threshold (10^{-6} is used), the candidate piece is considered to be an exact match, and the punishment weight w_e is subtracted from the fitness. A threshold is used rather than checking if the distance is exactly 0 because floating point arithmetic is used in calculating fitness values; therefore, rounding error may occur.

4.4 Genetic Operations

4.4.1 Selection

The selection scheme used by MICA is known as roulette wheel selection. This scheme is described in Section 2.3.

4.4.2 Crossover

Once two candidate pieces are selected, a crossover operation is performed with probability p_c. The value of p_c is adjustable, but a typical value MICA uses for p_c is 0.6. The crossover operation provides a means of combining two candidate pieces in the hopes of creating a more fit piece as a result.

Two types of crossover are employed by MICA: (1) single-point crossover, and (2) double-point crossover. Both crossover types have an equal probability of occurring once MICA decides to perform a crossover, and the two types are mutually exclusive for a given pair of candidate pieces. In other words, if one type of crossover is performed on a pair of candidate pieces, the other is not.

Single-point crossover is illustrated in Fig. 4.5. It begins by randomly selecting a crossover point p. The point p is the beginning index of a measure between the second measure and the last measure. Recall that every candidate piece has the same metric structure for a given execution of the program, so neither piece's length will be altered. After p is selected, the program checks if the DTPoint directly before p in each piece has a valid Markov transition to the point at p in the other piece. If and only if this is the case, the crossover is performed.

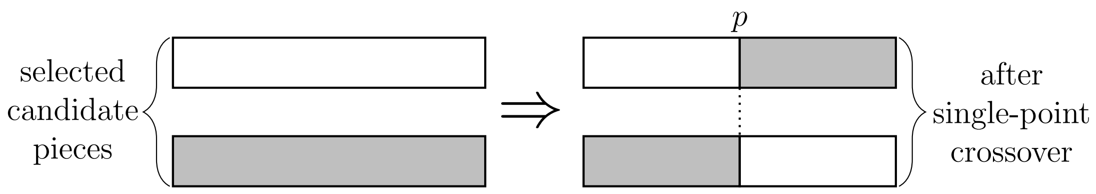


Figure 4.5: Single-point crossover. The crossover point p is randomly selected.

The two-point crossover is illustrated in Fig. 4.6. Two crossover points, p and q are randomly selected. The program ensures that $p < q$. As with the single-point crossover, the crossover points can occur only at measure start points, excluding the first measure. Again, the program ensures that the new transitions that arise from the crossover are valid. Specifically, the transition from the DTPoint before p in one piece to the point at p in the other piece, and the DTPoint before q in one piece to the point at q in the other piece must be valid transitions in the Markov transition matrix, otherwise the crossover is abandoned.

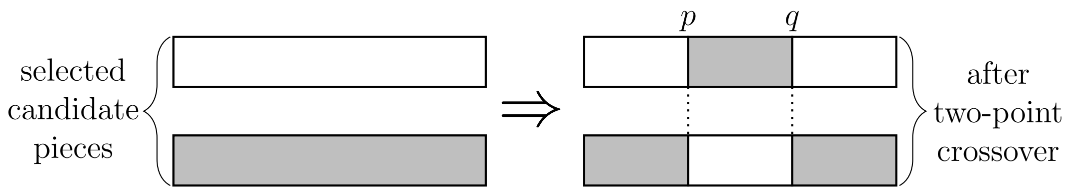


Figure 4.6: Two-point crossover. The crossover points p and q are randomly selected.

4.4.3 Mutation

The mutation operation stochastically changes the content of a candidate solution. In MICA, there are two mutation operations: (1) the measure regeneration mutation, and (2) the measure swap mutation.

In the measure regeneration mutation, each measure of a candidate piece is attempted to be regenerated randomly with a probability p_r. A typical value for p_r in MICA is 0.2. A measure is regenerated by traversing the Markov transition matrix beginning with the DTPoint from the previous measure. In the case of the first measure, the mutation begins by choosing an appropriate initial DTPoint at the correct metric position (since the initial metric position may vary with pick-up measures of different lengths). Each attempt to regenerate a measure is only accepted if the final DTPoint in the newly generated measure contains a valid transition into the first DTPoint of the next measure. This ensures that every candidate piece stays within the music syntax set forth by the Markov transition matrix.

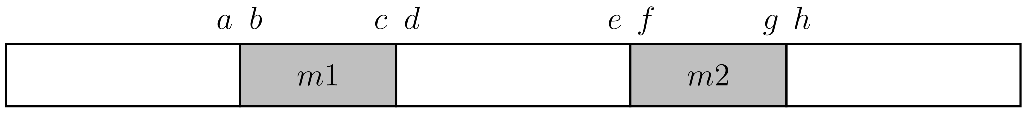
The measure swap mutation randomly selects two measures, which are neither the first nor the last measure, and attempts to swap those two measures. This mutation occurs with probability p_s. A typical value for p_s in MICA is 0.4. The mutation is only accepted if all the new DTPoint transitions that arise are valid. The DTPoints relevant to these transitions are illustrated in Fig. 4.7. The grey sections, $m1$ and $m2$, represent the measures to be swapped. For a measure swap mutation wherein the two measures to be swapped are not adjacent to each other, the following transitions must be valid in the Markov transition matrix:

$$a \to f,\ c \to h,\ e \to b,\ g \to d.$$

When the two measures to be swapped are adjacent to each other, the following set of transitions must be valid in the Markov transition matrix:

$$a \to f,\ g \to b,\ c \to h.$$

If and only if these transitions are valid for an attempted swap mutation, the mutation is finalized. Otherwise the mutation is abandoned.



Point	Description
a	DTPoint directly before $m1$
b	First DTPoint of $m1$
c	Last DTpoint of $m1$
d	DTPoint directly after $m1$
e	DTPoint directly before $m2$
f	First DTPoint of $m2$
g	Last DTpoint of $m2$
h	DTPoint directly after $m2$

Figure 4.7: Points of interest in the swap mutation.

4.4.4 Elitism

Elitism is the mechanism that copies the two most fit solutions from the population at time t into the next population. This simply ensures that the two best solutions are never lost via crossover or mutation. As a result of elitism, the best fitness of the current population is guaranteed to increase monotonically from population to population.

4.5 Termination Condition

The algorithm is terminated once the fitness of the best candidate piece converges to a consistent level. This is determined when the fitness of the best candidate goes unchanged for a number of generations. For example, after 20 generations with no fitness improvement, the GA terminates and returns the most fit solution.

4.6 Selecting a New Key

Each musical key is said to have a "key character;" an inherent set of features that are associated with a given key (Purwins, 2005). Russian composer Alexander Scriabin believed that each key had an associated colour (Vernon, 1942). However, the explicit relationship between key-invariant musical content and the optimal key for that content is by no means clear, if such a relationship exists at all. Therefore, MICA makes no attempt to choose a key based on the musical content produced. However, to add variability to MICA's output, a key for a new output piece is randomly selected uniformly across the range of keys from the input training set.

Transposing the key of a piece programatically is a trivial matter. The program needs only to add or subtract a given number of semi-tones to every pitch value in the piece. Deciding which key is most appropriate for a given piece is a much more subtle and ill-defined procedure. If the user wishes for an output piece to be in a different key, he or she easily can instruct the software to perform this transposition.

4.7 Implementation Details

The algorithm runs entirely within PWGL, a music programming environment based on the Lispworks implementation of Common Lisp. The representation of scores in PWGL is accomplished using the Expressive Notation Package (ENP), which allows the user to graphically edit and playback scores.

The training pieces are stored on the user's drive as ENP files. The files are created by importing a MIDI file, obtained from either *The Classical Archives* (`http://www.classicalarchives.com/`) or *The Mutopia Project* (`http://www.mutopiaproject.org/`), then making any necessary edits in PWGL's Score Editor to ensure the score is correct.

From these ENP files, MICA creates scores in the standard representation of Section 3.1, and the discrete-time representation of Section 3.2. MICA can also convert a score back into ENP format. This is particularly useful when the user wishes to visualize or hear a score that MICA produced.

An example PWGL patch for running MICA is shown in Fig. 4.8. The "get-enp-directory" box prompts the user to specify the folder in which the ENP files for the training set reside. This folder also may be specified by passing the path as a string. The "make-enp-score-db" box creates an encapsulation of the group of ENP files called an enp-score-db object, which is essentially an array of the ENP files in memory. This enp-score-db object serves as one input to the "make-mica-algorithm" box. The other input comes from the "make-mica-parameters" box which specifies all of the heuristic parameters for MICA, such as mutation and crossover probabilities, the weights of each Zipf metric, etc. The "make-mica-algorithm" box performs all of the preprocessing on the training set, and prepares the fitness function. The "compose-new-piece" box randomly selects a skeleton piece and runs MICA's GA to compose a new piece of music. The remaining boxes convert MICA's output back into an ENP string and create a visual interactive score in the Score-Editor at the bottom of the patch.

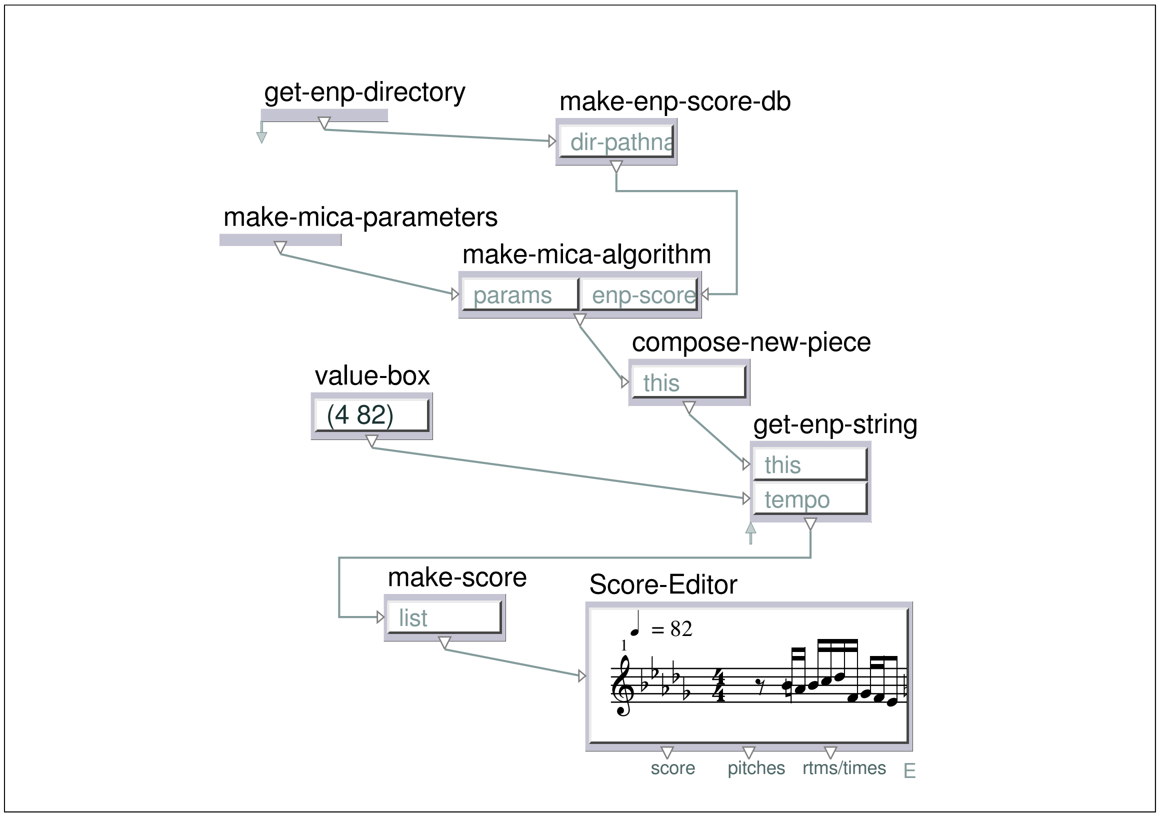


Figure 4.8: An example PWGL patch for running MICA.

4.8 Abandoned Methods

A central theme in corpus-based algorithmic composition is balancing the tradeoff between the algorithm's creative freedom, and the stylistic accuracy of its output.

Determining the extent to which the output may differ from training pieces while still remaining stylistically representative of the training pieces is crucial to the success of the algorithm.

Several approaches were attempted prior to settling on the method presented earlier in this chapter. All of these approaches involved widening the GA's search space by permitting the algorithm more freedom to explore a wider range of musical sequences, rather than restricting the output to valid Markov sequences using the discrete-time representation from Section 3.2. This difference in creative freedom was manifested at the initial mating pool generation stage, and the mutation stage of the algorithm.

The alternative mating pool generation methods attempted were as follows:

1. Generate rhythms in each voice using a probability mass function (PMF) of all rhythm values in each respective voice of the training pieces, then assign pitches to the rhythm values using a PMF of all pitches in each respective voice.

2. Generate the rhythm using a Markov model of the discrete time representation that ignores pitch values, then assign pitches to using a PMF as in method (1).

3. Generate the rhythm using a Markov model of the discrete time representation that ignores pitch values as in method (2), then assign pitches using a Markov model of the pitch values in their standard score representation in each voice.

The mutation function used with these methods performed the following mutation operations randomly:

i. Alter a pitch value randomly.

ii. Split a note into two notes each with half the original rhythmic duration, assign a random pitch to the second note.

iii. Merge two adjacent notes into one.

iv. Convert a pitched note to a rest.

v. Convert a rest to a random pitch value.

vi. Swap two notes in a measure, both rhythm and pitch.

vii. Reverse the order of notes in a measure.

viii. Regenerate a measure.

ix. Swap two measures in a voice.

These mutations were performed in the standard representation of scores, instead of the discrete-time representation. In the case of a Markov model being used to generate rhythm or pitch values instead of a PMF, as with generation methods (2) and (3), the mutations would ensure that every note in the piece remains within a valid sequence generated by that Markov model.

Using the aforementioned mating pool generation methods in conjunction with the mutation function, the algorithm was unable to generate any musically convincing output. Perhaps some of these methods would be successful with a different fitness function and mutation function, in particular the rhythm generation method used in methods (2) and (3).

For the tested training sets, the more restrictive method of using a Markov model of rhythm and pitch simultaneously in the discrete-time score representation provides a good balance between creative freedom and stylistic accuracy.

Chapter 5

Example Training Sets

5.1 Three Schumann Pieces From Album für die Jugend

The first training set consists of three simple pieces from Schumann's Op. 68, Album für die Jugend (Album for the Young):

1. No. 1, *Melodie (Melody)*
2. No. 3, *Trällerliedchen (Humming song)*
3. No. 5, *Stückchen (A little piece)*

These pieces are not technically demanding nor are they musically deep, as they are intended for children or novice pianists. The three pieces have a high degree of similarity, and provide a simple test bed for MICA's functionality.

All three of the pieces are in C-major, common time, and in two voices. Each piece features a simple melody in the upper voice consisting primarily of quarter notes, alongside a bass line in the lower voice consisting of eighth notes that are usually an octave plus a diatonic third below the melody on the beat, with a pedal tone interposed between each beat, usually the fifth degree of the scale (a G in C-major).

A good indicator of the commonality between each piece in a given training set is the distribution of the number of edges out of each state in the Markov transition matrix of DTPoint objects. The more edges out each state has, the more opportunities MICA will have to branch off into new sequences that combine small sections of training pieces with each other, or with different small sections of themselves.

The distribution of the number of edges out of every state in the transition matrix for the three Schumann pieces in pictured in Fig. 5.1. The mean number of possible transitions out of each node, $E[N_e]$, is

$$\begin{aligned} E[N_e] &= \frac{1 \cdot 118 + 2 \cdot 25 + 3 \cdot 5 + 4 \cdot 5 + 6 \cdot 1}{118 + 25 + 5 + 5 + 1} \\ &= \frac{209}{154} \\ &\approx \mathbf{1.36} \end{aligned}$$

Using this mean, the approximate size of the search space can be calculated. In general, the size search space depends on the skeleton piece selected, since each training piece may begin at a different metric position, and may have a different length. In the case of the Schumann training set, two pieces start at metric position 0, the other starts at metric position 4. Since there are 3 pieces, there are 3 potential

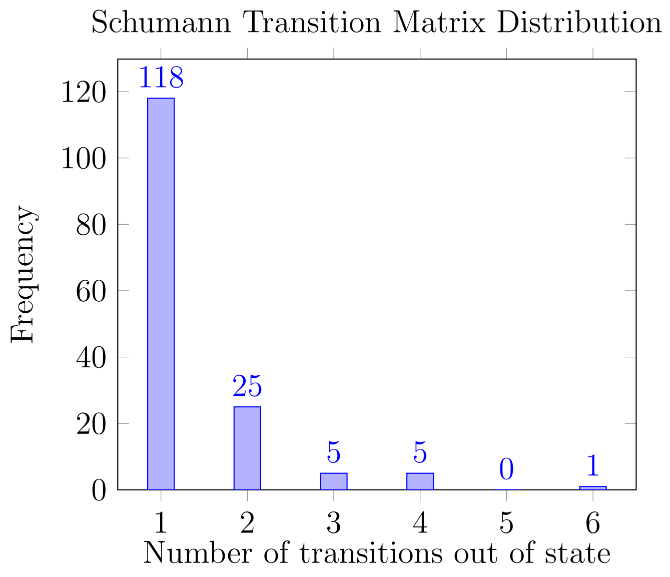


Figure 5.1: Distrbution of edges out of each state in the transition matrix for the Schumann training set.

final measures for a new piece.

In general, the size of the search space, $|S|$, may be estimated as

$$|S| = \dim(\pi_{\mathbf{0}}(i))\,\{E[N_e]\}^{l-1}\,N_f \tag{5.1}$$

where $\dim(\pi_{\mathbf{0}}(i))$ is the number of non-zero elements in the initial state distribution for the beginning metric position of the selected skeleton piece i, $E[N_e]$ is the mean number of transitions out of each node, l is the length of the the skeleton piece in DTPoints up to but not including the last measure, and N_f is the number of final measures in the training set (which is equal to the number of training pieces).

In the Schumann training set, 2 pieces begin at metric position 0 with the exact same DTPoint, and the third piece begins at metric position 4. Therefore,

$\dim(\pi_{\mathbf{0}}(i)) = 1$ for any selected skeleton piece in the Schumann training set. Assuming *Melodie* is the skeleton piece, and since the subdivision for the Schumann training set as calculated using Eq. 3.1 is 8, and *Melodie* is 24 measures long with the repeat sign unfolded, $l = 184$. As calculated from the distribution of transitions out of each node in Fig. 5.1, $E[N_e] \approx 1.36$. The number of final measures, N_f, is 3. Using Eq. 5.1, the size of the GA's search space for the Schumann training set is

$$\begin{aligned} |S| &= \dim(\pi_{\mathbf{0}}(i)) \left\{E[N_e]\right\}^{l-1} N_f \\ &= 1 \cdot 1.36^{183} \cdot 3 \\ &\approx \mathbf{8.23 \cdot 10^{24}} \end{aligned}$$

This is the cardinality of the set of all possible candidate pieces in which the GA must search for a good solution according to the fitness function.

The progression of the fitness values for a sample execution of MICA using the Schumann training set is shown in Fig. 5.2.

An example MICA composition using the Schumann training set is provided in Appendix A.1.

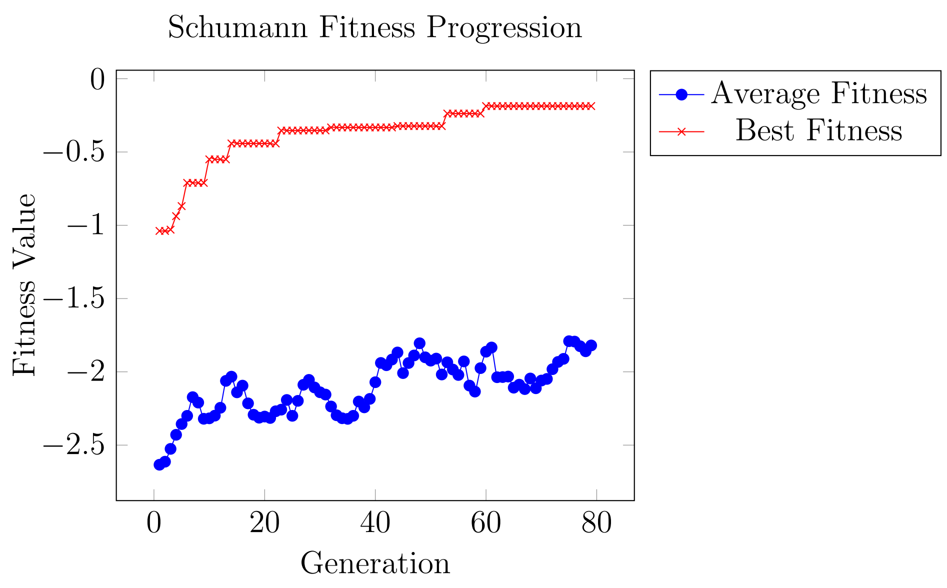


Figure 5.2: The progression of the fitness values for a sample execution of MICA using the Schumann training set. For this run of the algorithm, the mating pool size was 80 chromosomes.

5.2 Five Bach Two-Part Inventions

Five of Bach's Two-Part comprise the second example training set, which is significantly more complex musically than the *Album für die Jugend* training set. These inventions are:

1. Invention No. 2 in C minor, BWV 773
2. Invention No. 7 in E minor, BWV 778
3. Invention No. 11 in G minor, BWV 782
4. Invention No. 13 in A minor, BWV 784
5. Invention No. 15 in B minor, BWV 786

The BWV (Bach-Werke-Verzeichnis, translates to Bach Works Catalogue) numbers are catalogue numbers that are used to document and organize Bach's massive corpus of works. All five of these inventions are in minor keys, and in $\frac{4}{4}$ time. Each invention is an archetypal example of Baroque two-part counterpoint. Ornaments such as trills and mordents are written out explicitly in the training data. The subdivision as calculated using Eq. 3.1 is 32 for the Two-Part Invention training set.

The distribution of transitions out of each state in the transition matrix for the Bach 2-Part Invention training set is pictured in Fig. 5.3.

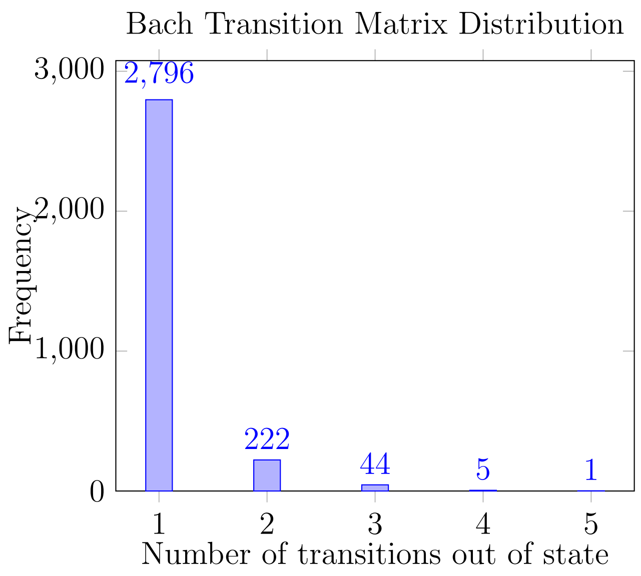


Figure 5.3: Distrbution of edges out of each state in the transition matrix for the Bach training set.

The mean number of possible transitions out of any given node in the Bach training set, $E[N_e]$, is

$$\begin{aligned} E[N_e] &= \frac{1 \cdot 2796 + 2 \cdot 222 + 3 \cdot 44 + 4 \cdot 5 + 5 \cdot 1}{2796 + 222 + 44 + 5 + 1} \\ &= \frac{3397}{3068} \\ &\approx \mathbf{1.11} \end{aligned}$$

In the Bach training set, all pieces begin at metric position 0, and 4 out of 5 pieces have an identical initial state with the exact same DTPoint. Therefore, $\dim(\pi_{\mathbf{0}}(i)) = 2$ for any selected skeleton piece in the Bach training set. Assuming Invention No. 15 is the skeleton piece, and since the subdivision for the Bach training set is 32, and Invention No. 15 is 22 measures long, $l = 672$. As calculated from the distribution of transitions out of each node in Fig. 5.3, $E[N_e] \approx 1.11$. The number of final measures, N_f, is 5. Using Eq. 5.1, the size of the GA's search space for the Bach training set is

$$\begin{aligned} |S| &= \dim(\pi_{\mathbf{0}}(i)) \left\{E[N_e]\right\}^{l-1} N_f \\ &= 2 \cdot 1.11^{671} \cdot 5 \\ &\approx \mathbf{2.58 \cdot 10^{31}} \end{aligned}$$

The way in which MICA combines sections of training pieces to form new pieces is pictured in Fig. 5.4. Pictured are the first two measures from a MICA composition, along with the training pieces from which each note in the new composition originated. Every enclosed section is necessarily overlapping, due to the fact that the composition was generated from a first order Markov transition matrix. Note that sometimes the overlap spans more than one note. The analysis in Fig. 5.4 was done by hand, so it is indeed possible that some smaller subsections originated from more than one training piece, or small subsequences from one training piece may be present within a longer subsequence from another training piece. The program does not "know" which pieces it is combining at any given point in time, the combination happens as a result of stochastically traversing the Markov transition matrix.

The progression of the fitness values for a sample execution of MICA using the Bach training set is shown in Fig. 5.5.

An example MICA composition using the Bach training set is provided in Appendix A.2.

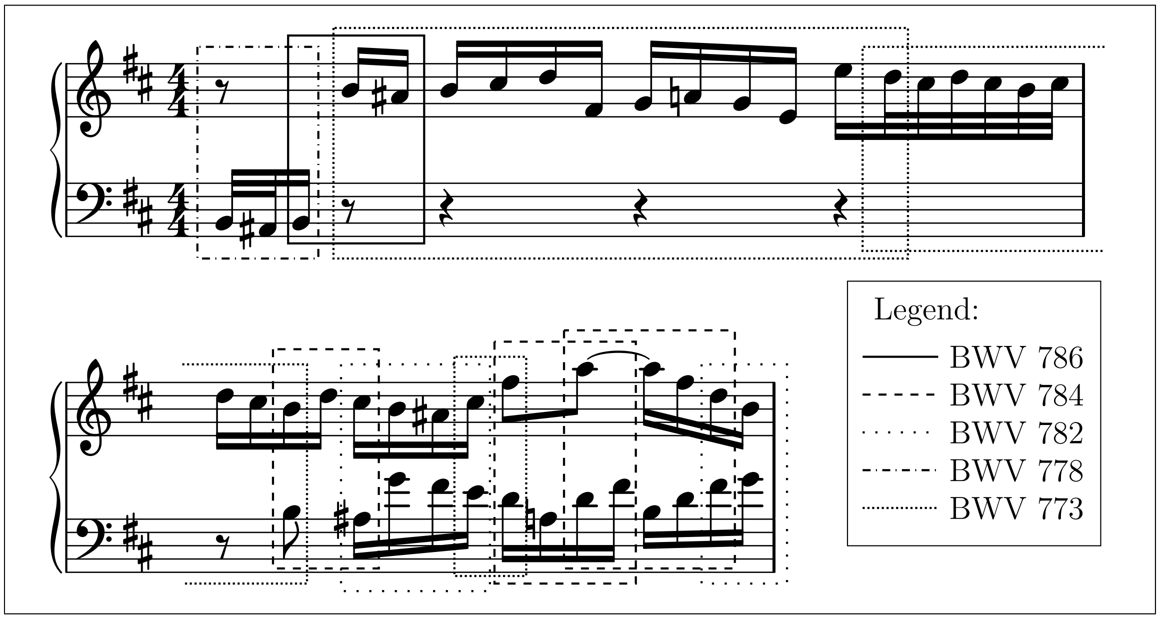


Figure 5.4: Analysis of the pieces form which small sections of a MICA composition are produced.

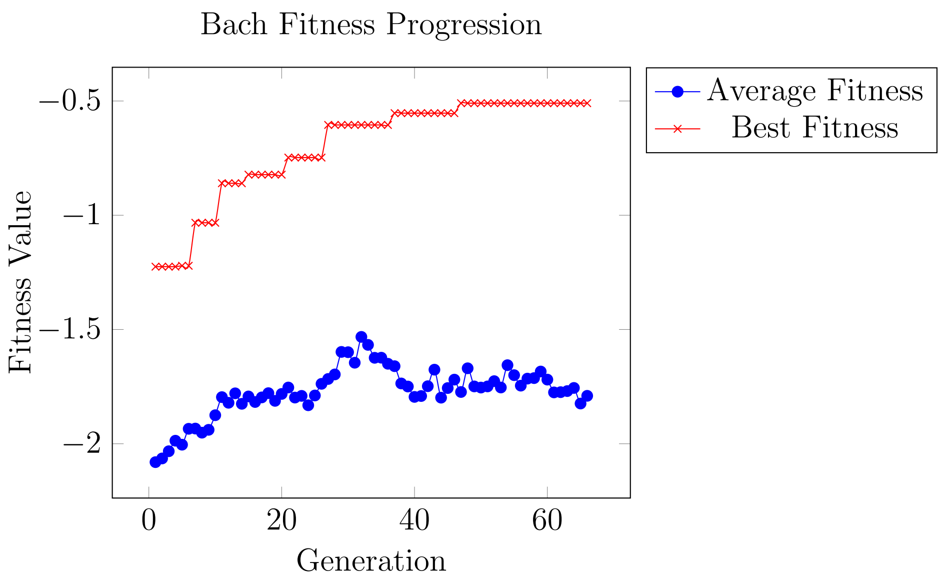


Figure 5.5: The progression of the fitness values for a sample execution of MICA using the Bach training set. For this run of the algorithm, the mating pool size was 80 chromosomes.

5.3 Five Mozart Menuettos from String Quartets

This training set illustrates a current limitation of MICA. Despite containing five String quartets that sound quite similar musically, MICA is unable to find sufficient commonality within the training set to produce compositions that vary enough from the training pieces.

The distribution of edges out of each DTPoint in Fig. 5.6 shows that a very high percentage of states in the transition matrix have only one possible next state. The mean number of possible transitions out of any given node in the Mozart String

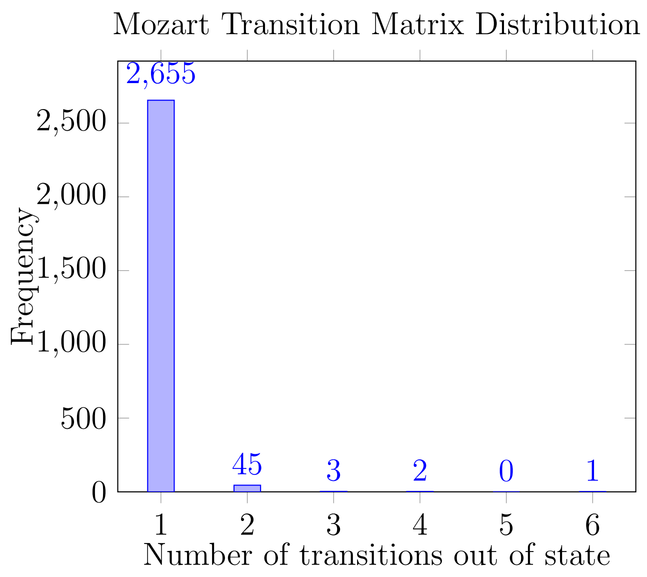


Figure 5.6: Distrbution of edges out of each state in the transition matrix for the Mozart training set.

Quartet training set, $E[N_e]$, is

$$\begin{aligned} E[N_e] &= \frac{1 \cdot 2655 + 2 \cdot 45 + 3 \cdot 3 + 4 \cdot 2 + 6 \cdot 1}{2655 + 45 + 3 + 2 + 1} \\ &= \frac{2768}{2701} \\ &\approx \mathbf{1.02} \end{aligned}$$

The value of $E[N_e]$ being so close to 1 implies that traversing the transition matrix will reproduce long sections that are identical to sections from the training set. Usually these sections are 4-5 measures long, or more. Directly copied sections of this length are probably too long for the output to be considered a "new" composition. Therefore, no output examples are provided for this training set. A possible method for rectifying this problem is discussed in Section 8, Future Work.

5.4 Execution Time

This section provides some example execution times for MICA. All benchmarking tests were performed on a MacBook Air with a 1.7 GHz Intel Core i7 processor and an 8 GB 1600 Mhz DDR3 memory, running OS X Version 10.8.5.

The preprocessing phase of MICA is entirely deterministic (refer to Fig. 4.1). Therefore, the execution time for the preprocessing phase is essentially fixed for a given training set. Table 5.1 summarizes the execution time for the preprocessing phase in the Schumann and Bach training sets.

A summary of three example execution times for GA phase of the Bach and Schumann training sets is shown in Table 5.2. The execution time is a function of the size of the transition matrix, the number of generations in the GA, and the many other random processes within the GA.

Table 5.1: The preprocessing execution time for the Bach and Schumann training sets.

Training Set	**Execution Time (seconds)**
Schumann	0.528
Bach	5.434

Table 5.2: The GA execution time for the Bach and Schumann training sets.

Training Set	**Execution Time (minutes)**	**Generations**	**Skeleton Piece**
Schumann	0:49.029	46	*Stückchen*
	1:12.924	71	*Melodie*
	1:27.429	87	*Melodie*
Bach	2:56.907	60	BWV 778
	4:13.954	79	BWV 773
	4:23.515	95	BWV 784

Chapter 6

Human Evaluation

A group of volunteers were gathered to evaluate the musicality of MICA's output, alongside actual historical compositions by Bach and Schumann as control pieces. To help avoid bias, the volunteers were not informed about the nature of the music they were evaluating, and the volunteer's data was collected via an online HTML survey. Two groups of students participated in the survey: (1) seven Music students, and (2) four Engineering students. This is a relatively small sample size, so we must acknowledge the limitation of this data set, and exercise caution when making any inferences from the data. In a single survey, the subject heard a total of four pieces. Each of the four pieces was randomly selected at page load from each of the following groups:

A. Four MICA-composed pieces using the Schumann training set.

B. Four MICA-composed piece using the Bach training set.

C. Four actual Schumann compositions.

D. Four actual Bach compositions.

Every survey contained one piece from every group A through D. The order in which the pieces were played back was randomized at every page load to help eliminate bias that might stem from the order of playback. The four compositions used in group C were all pieces from Schumann's *Album für die Jugend*, namely:

i. No. 2, *Soldatenmarsch (Soldiers' march)*

ii. No. 7, *Jägerliedchen (Hunting song)*

iii. No. 8, *Wilder Reiter (The wild rider)*

iv. No. 18, *Schnitterliedchen (The reaper's song)*

The four compositions used in group D were Bach keyboard works in two voices, specifically:

i. *Bourrée I* from English Suite No. 1 (BWV 806)

ii. *Allemande* from French Suite No. 3 (BWV 814)

iii. *Allemande* from French Suite No. 5 (BWV 816)

iv. *Bourrée* from French Suite No. 5 (BWV 816)

Of the MICA-composed pieces in groups A and B, half were evolved using only the harmonic interval metric for the subdivided Zipf metrics in the fitness function, and the other half were evolved both the harmonic interval and pitch distance metrics for the subdivided Zipf metrics in the fitness function. This was done to help indicate whether the use of more subdivided Zipf metrics produces a more musical and stylistically correct output.

After hearing a piece, subjects were asked the following questions:

- Which composer (if any) does this sound like to you?
- Have you heard this piece before?
- Rate the musicality of this composition from 1 to 10, with 1 being the least musical, and 10 being the most musical.
- Leave any thoughts or comments you have about the compositional aspects of the piece.

The volunteers' musicality ratings for each group of pieces is shown in Table 6.1. In general, the Engineering students were inclined to give higher ratings than the Music students.

Table 6.1: Volunteers' musicality ratings for each type of piece.

Volunteer type	**Piece**	**Musicality ratings**	**Avg.**	**Std. dev.**
Music students	Actual Schumann	3, 4, 5, 6, 7, 8, 9	6	2.16
	Actual Bach	5, 5, 8, 8, 8, 9, 9	7.42	1.72
	MICA's Schumann	2, 2, 3, 5, 5, 7, 7	4.42	2.15
	MICA's Bach	3, 6, 7, 8, 8, 9, 9	7.14	2.12
Eng. students	Actual Schumann	4, 7, 8, 8	6.75	1.89
	Actual Bach	6, 7, 7, 10	7.5	1.73
	MICA's Schumann	3, 6, 8, 9	6.5	2.65
	MICA's Bach	8, 8, 9, 10	8.75	0.96
All	Actual Schumann	3, 4, 4, 5, 6, 7, 7, 8, 8, 8, 9	6.27	2.00
	Actual Bach	5, 5, 6, 7, 7, 8, 8, 8, 9, 9, 10	7.45	1.63
	MICA's Schumann	2, 2, 3, 3, 5, 5, 6, 7, 7, 8, 9	5.18	2.44
	MICA's Bach	3, 6, 7, 8, 8, 8, 8, 9, 9, 9, 10	7.73	1.90

The musicality ratings for the two different subdivided metrics are shown in Table 6.2. For pieces evolved using only one subdivided Zipf metric, the harmonic interval metric was used. For pieces evolved using two subdivided Zipf metrics, the harmonic interval metric and the pitch distance metric were used. The results seem to indicate that the use of one subdivided metric produced a more musical output for the Schumann training set, and the use of two subdivided metrics produced a more musical output for the Bach training set.

Again, we must acknowledge the limitations of the small sample size. Since the surveys were generating randomly at page load, none of the Engineering students evaluated a MICA-composed Bach piece that was evolved using one metric by random chance.

A comparison of each individuals rating of MICA's against historic compositions is presented in Table 6.3. Interestingly, more volunteers rated the MICA-composed

Table 6.2: Volunteers' musicality ratings for the two different subdivided metrics.

Volunteer type	**Training Set**	**Subdivided Zipf metrics**	**Musicality ratings**	**Avg.**	**Std. dev.**
Music students	Schumann	One	5, 7	6	1.41
		Two	2, 2, 3, 5, 7	3.8	2.17
	Bach	One	3, 6, 8, 8, 9	6.8	2.39
		Two	7, 9	8	1.41
Eng. students	Schumann	One	6, 8, 9	7.67	1.53
		Two	3	3	N/A
	Bach	One	N/A	N/A	N/A
		Two	8, 8, 9, 10	8.75	0.96
All	Schumann	One	5, 6, 7, 8, 9	7	1.58
		Two	2, 2, 3, 3, 5, 7	3.67	1.97
	Bach	One	3, 6, 8, 8, 9	6.8	2.39
		Two	7, 8, 8, 9, 9, 10	8.5	1.05

Table 6.3: Individual Musicality Ratings of MICA-composed pieces vs. actual historic pieces.

Volunteer type	**MICA > Bach**	**MICA = Bach**	**MICA < Bach**	**MICA > Schumann**	**MICA = Schumann**	**MICA < Schumann**
Music students	1	4	2	1	2	4
Eng. students	3	1	0	1	1	2
All	4	5	2	2	3	6

Bach higher than historic Bach compositions. Schumann's historic compositions rated higher than the MICA-composed Schumann.

Neither the Music students nor Engineering students claimed to have heard any of the pieces before, with the exception of one Engineering student who claimed to have previously heard a MICA-composed Schumann piece.

Five out of seven Music students correctly identified the actual Bach compositions as sounding like Bach, and only one out of seven Music students identified the actual Schumann as sounding like Schumann.

Five out of seven Music students identified the MICA-composed Two-Part Inventions as sounding like Bach, while zero identified the MICA-composed Schumann as sounding like Schumann.

None of the Engineering students successfully identified the composer of any historic piece, or the training set composer of a MICA-composed piece.

In the additional comments section, the Engineering students wrote primarily of the emotions that each piece evoked ("happy", "angry", etc.), while the Music students focused on the compositional aspects of the music.

Some typical comments on the actual Schumann pieces from *Album für die Jugend*:

> "The melodies are not varied enough. Outlining of chords in this case is uninteresting."

> "This piece sounds as if it belongs to a dance suite of some variety. The form is easy to follow and the melody is not unpredictable."

The MICA-Composed Schumann was usually interpreted by the volunteers as a remedial piano exercise:

> "Sounds like a piano exercise. Very little musicality."
>
> "It sounds as if it serves for developing a single technique. Is it an étude of some variety?"
>
> "Really repetitive... no development or modulation. "
>
> "Really boring, it sounded like more of a musical exercise than a musical composition."

Surprisingly, the actual Bach pieces received negative, neutral, and positive comments:

> "The bass line was very stark and did not complement the melody well."
>
> "This piece sounds like it belongs to the late Baroque era. The harmonies and counterpoint sounds reflects the Baroque period."
>
> "There are many sequences in this piece."

The most interesting critical comments came with the computer composed Bach:

> "It lacked a lot of the intricacies of Bach that I am normally accustomed to - some of the lines seemed as though their contours did not deviate from a single model."
>
> "This piece sounds like it belongs to the romantic era as it uses a variety of compositional techniques to change the texture of the two voices. It also changes textures and ranges very frequently which makes me it comes from this period."
>
> "A nice variety of accompaniments. Melody is not entirely clear."
>
> "Sounds either Baroque or early Classical style. The counterpoint is indicative of this."

The observation the MICA-composed Bach lacks many of the “intricacies” of actual Bach is a fair critical observation. No rules are hard coded into MICA, such as the rules of Baroque counterpoint, so it is unlikely that MICA will produce a piece that follows counterpoint rules strictly. The tradeoff is MICA’s generality which comes without the explicit encoding of musical rules.

Chapter 7

Conclusion

The algorithm presented is able to synthesize new music that is stylistically representative of the input training pieces. Human evaluation of MICA's output indicates that the "musicality" of these compositions is on par with historic compositions for Schumann and Bach training sets in two voices, and the intended style of the computer compositions is recognizable by Music student volunteers. For training sets with higher numbers of voices, such as string quartets, MICA struggles to find sufficient commonality to produce new and interesting music. The representation of musical scores and the novel combination of techniques used for computer analysis and synthesis of musical compositions resulting from this work, form a successful general purpose style-imitation computer composer.

Chapter 8

Future Work

Although the human evaluation suggests MICA is fairly successful in its goal of stylistic algorithmic composition, room for improvement remains in certain aspects of the program. Currently, the input training set must have a very high degree of commonality for MICA to extract musical information that can be recombined in interesting ways. This is evidenced by MICA's inability to sufficiently recombine the Mozart String Quartet training set.

Much of the potential for improvement lies in the preprocessing stages. An algorithm for intelligently changing a minor piece to major, and vice versa, would be useful. The program could then use a much greater portion of a composer's works for a certain training set. Furthermore, by virtue of recomposing the piece to a different mode at the outset, long sequences directly from the recomposed piece might already sound "new."

Another way to increase MICA's ability to find commonalities is to add bar lines at every beat of the training data, effective converting every piece to $\frac{1}{4}$, or, depending on the beat value, $\frac{1}{8}$, $\frac{1}{16}$, etc. This would increase the average number of transitions out of every state in the transition matrix, and it carries the added bonus of allowing a wide range of time signatures in a single training set. Even 3-based and 2-based meters could be used within the same training set. The drawback of this method is that the significance of metric position is largely lost. The GA would have a larger search space, but perhaps the fitness function would need to account for the metric position of notes in some way.

Some adjustments may also be made to the genetic algorithm itself to improve the efficiency of the optimization process. The mutation function currently attempts to mutate the piece by regenerating each measure probabilistically. Instead, it may be more effective to attempt to regenerate sequences of random length, rather than restricting the regeneration to the bar lines. An adaptive crossover and mutation scheme, as outlined in (Srinivas and Patnaik, 1994), could also increase the efficiency of the optimization process.

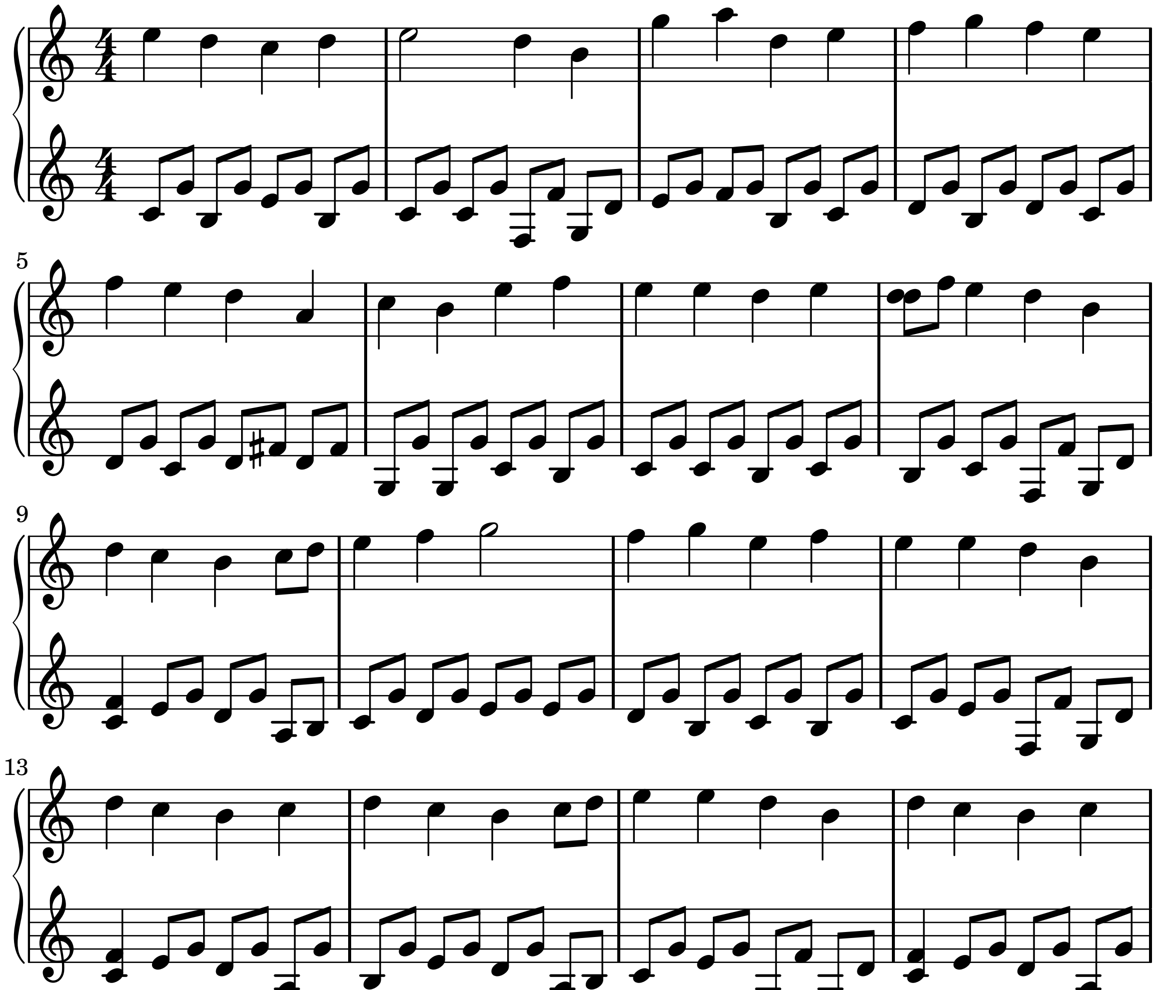
Appendix A

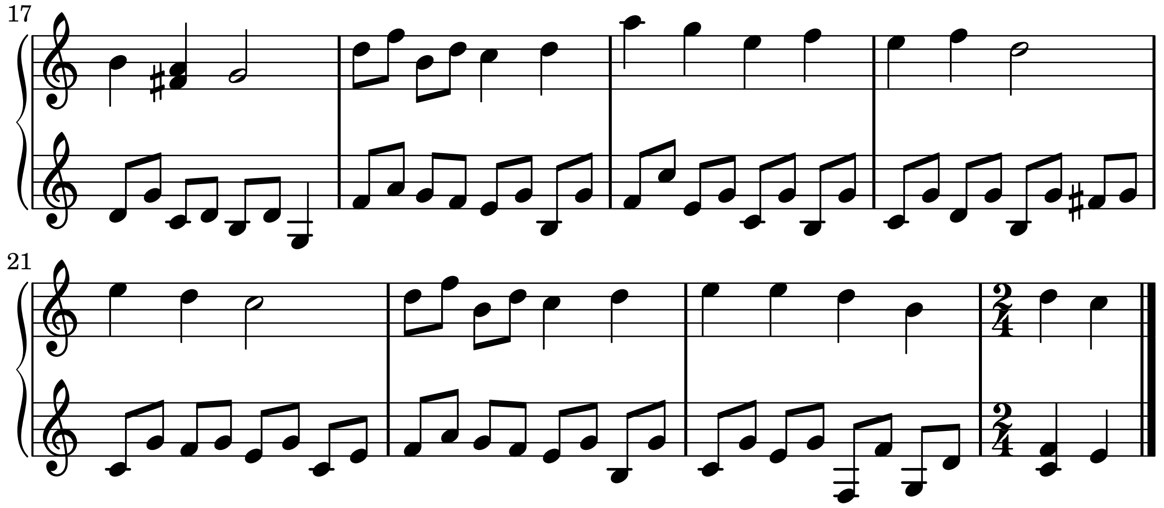
MICA Compositions

Some examples of MICA's output are presented in this Appendix. The format of MICA's output is ENP files, the native PWGL format for musical scores. These files were converted to Lilypond files for readability.

A.1 Schumann Training Set Example Score

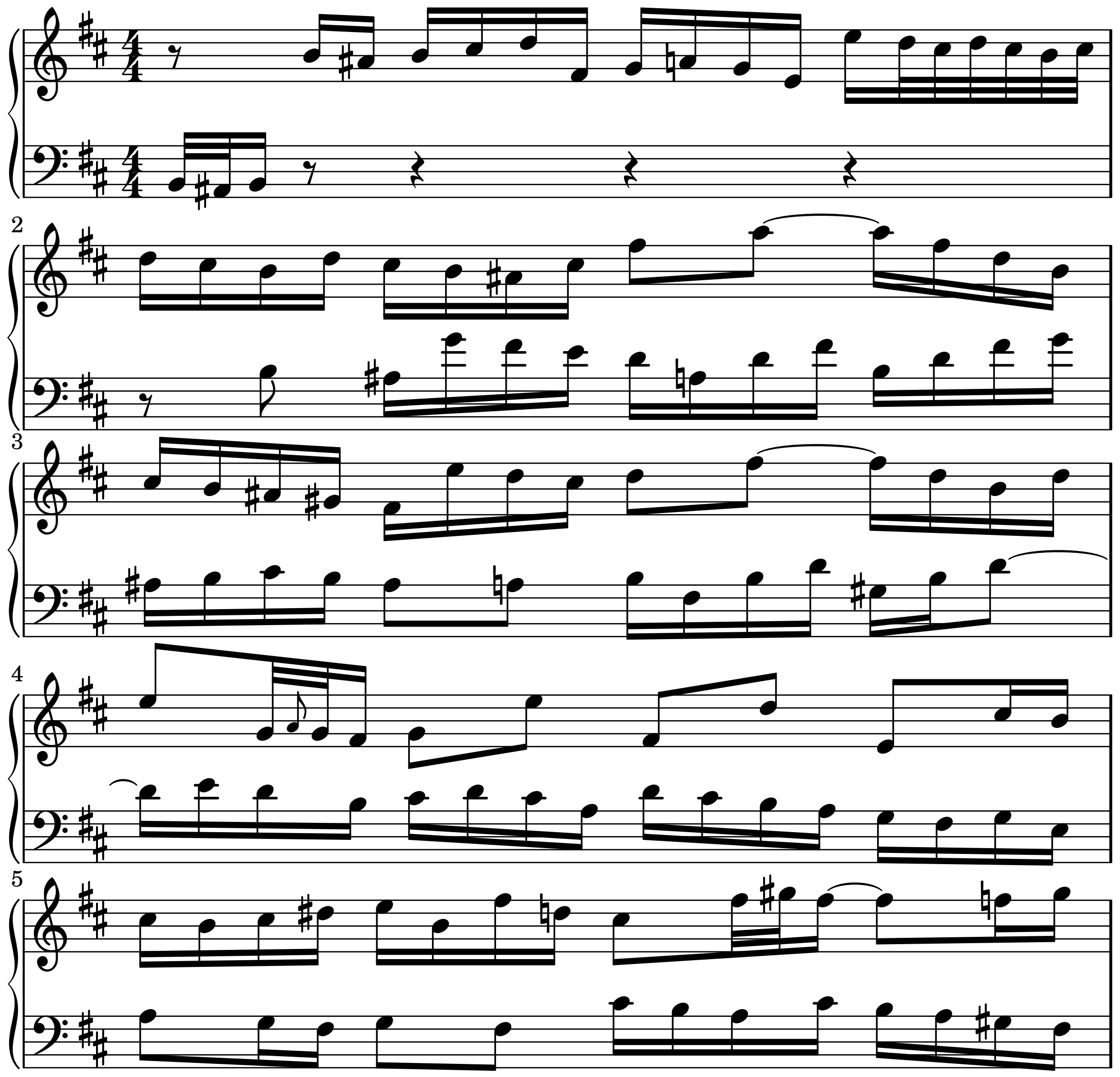
The score in this section is an example of MICA's output using the Schumann *Album für die Jugend* training set described in Section 5.1. Note that the final measure is in $\frac{2}{4}$ time because it was taken from the training piece with a pickup measure, despite the fact that the new composition does not have a pickup measure.

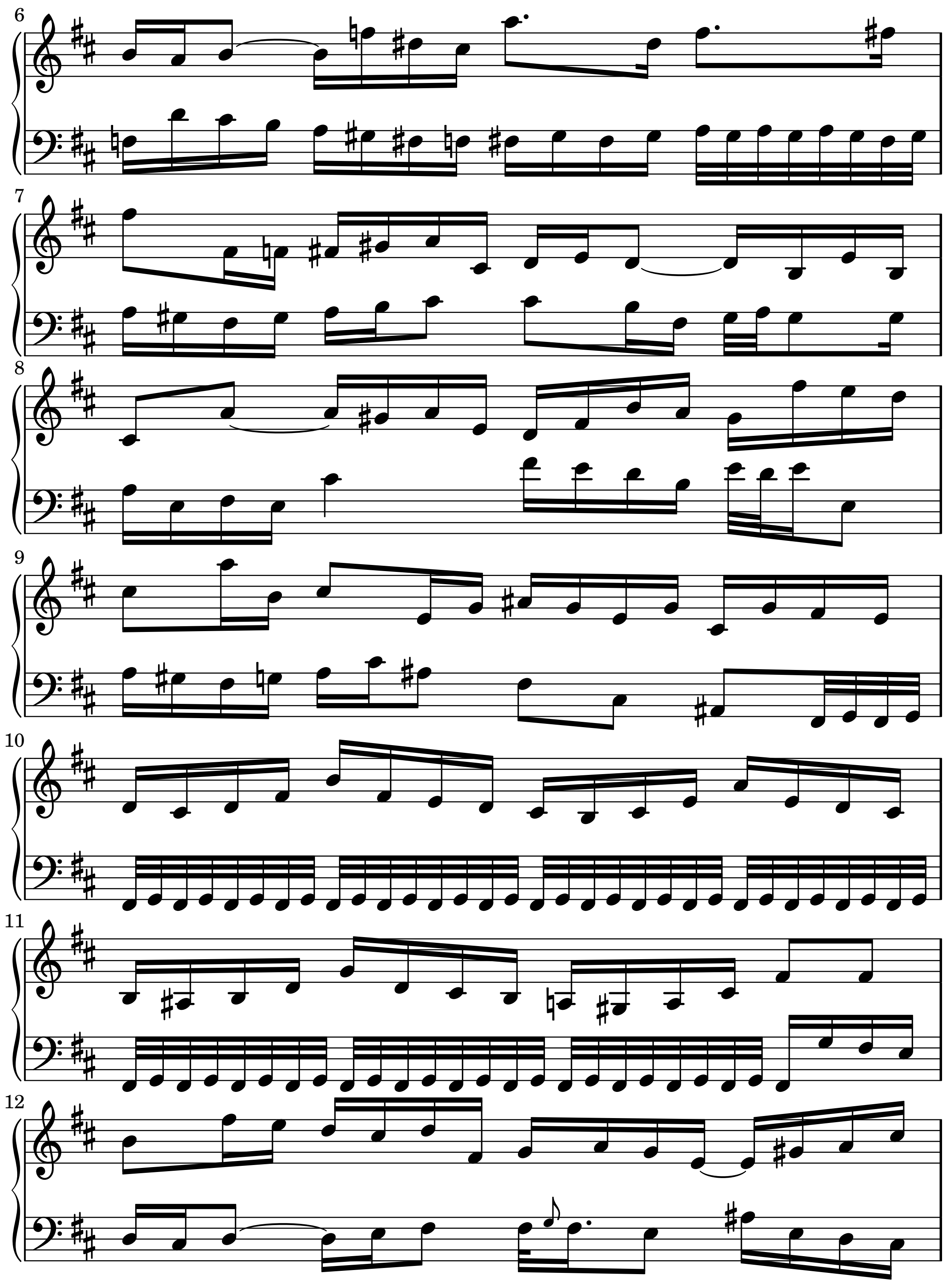


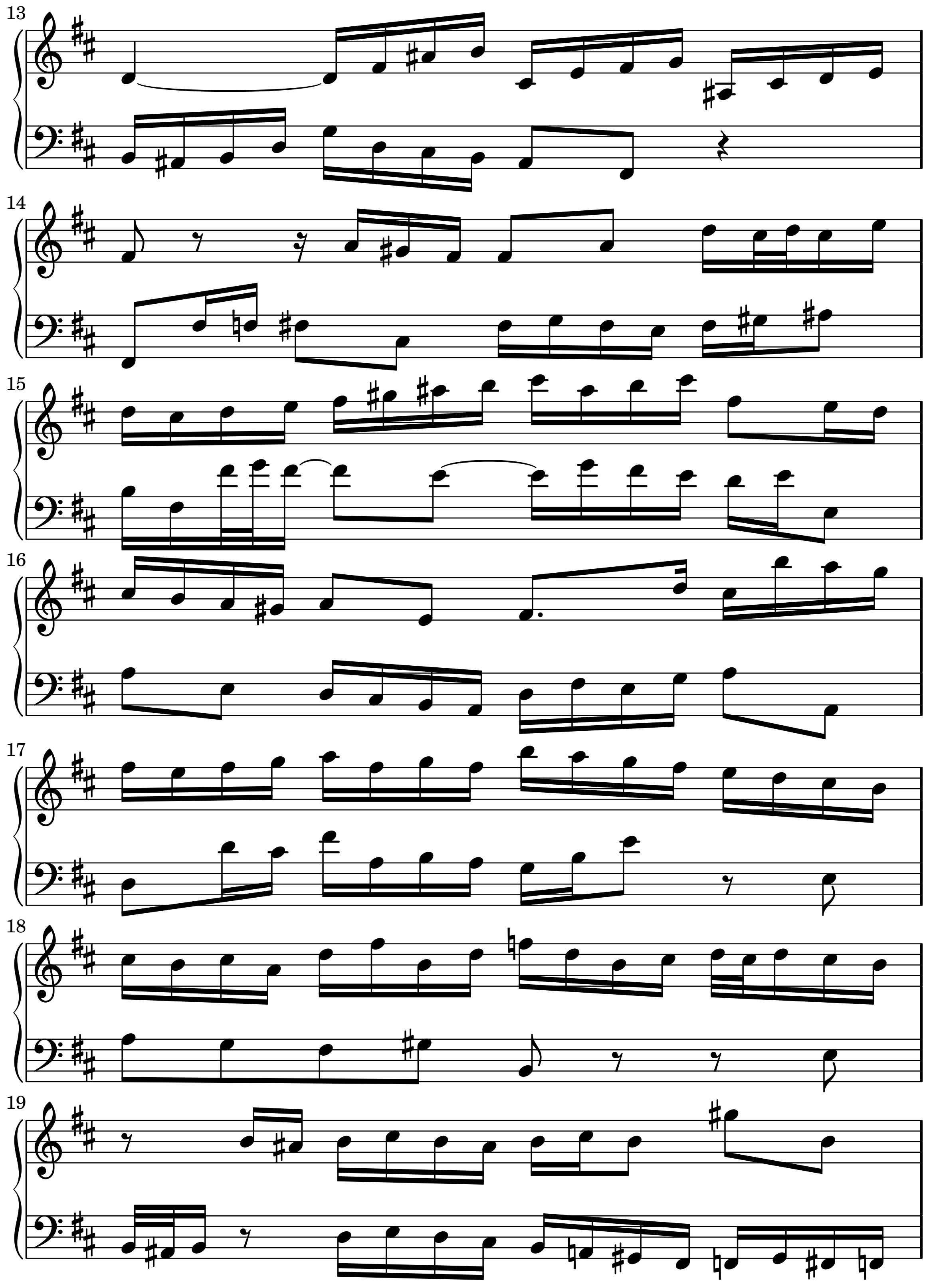


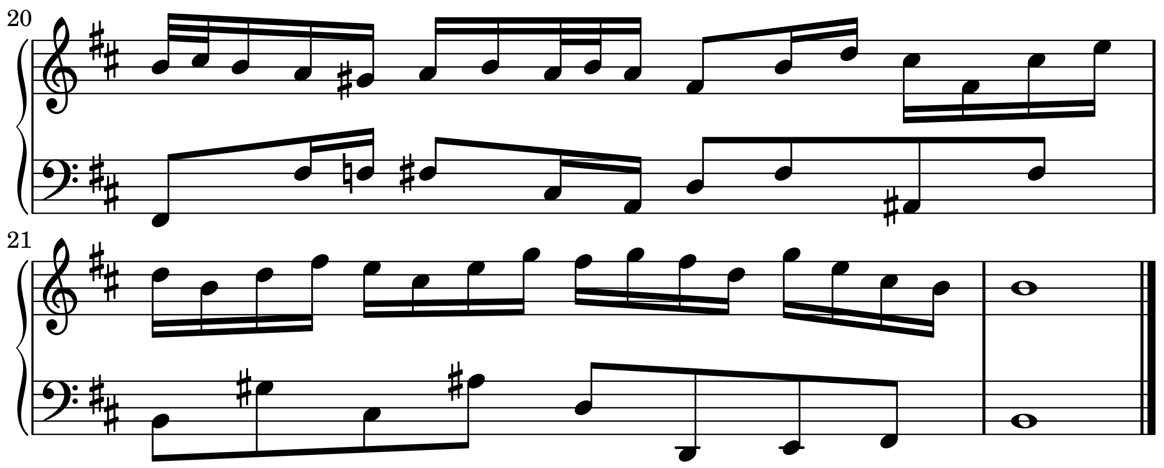
A.2 Bach Training Set Example Score

The score in this section is an example of MICA's output using the Bach Two-Part Invention training set described in Section 5.2. Trills are written out explicitly in this piece.




6
7
8
9
10
11
12


13
14
15
16
17
18
19


20
21

Glossary

accidental A note in a piece of music that is outside of the key signature. 26

counterpoint A style of music composition that uses multiple voices that are independent rhythmically, but interdependent harmonically. 50

enharmonic major Defined with respect to a minor key, the enharmonic major is the major key with the same root note as the minor key. For example, C major is the enharmonic major of C minor. 26

first and second ending A first ending is the end of a repeated section that should be played during the first pass, and the second ending should be played during the second pass. Used in conjunction with repeat signs. 3

half step See semi-tone. 25

harmonic interval In the context of MICA, the difference (in semi-tones) between two simultaneously played pitches. 29

Lilypond An open source music engraving program and file format. 66

measure A segment of time in a piece of music, whose duration is determined by the time signature. A measure is also known as a *bar*. In music notation, measures are separated by vertical lines called *bar lines*. 2

melodic interval In the context of MICA, the difference (in semi-tones) between two successive pitches in a voice. If the voice contains chords, the highest note in the chord is considered by MICA to be the melody note. 10

metric position Position of a note or rest within a measure. 3

MIDI Musical Instrument Digital Interface - a standard protocol for communication between digital music devices. A MIDI file is a standard digital representation of a music performance or music score. 3

MIDI note number A standard mapping of 128 notes to integer values. Middle C has a value of 60, and a semi-tone up or down is represented by a value of 1. For example, an octave above middle C is 72, and an octave below middle C is 48. 16

mode In music, the mode refers the type of scale used by the piece. In MICA, every training piece is assumed to be in either a major or minor mode, as most common practice era music is in one of these two modes. 25, 26

mordent A single quick alternation between a written note, and an adjacent note. 50

ornament In music, an ornament is a musical flourish added to a melody, such as a trill or a mordent. Ornaments are notated using special symbols, and the precise nature of the ornament is often at the discretion of the performer to some extent. 50

pickup measure A measure at the beginning of a piece of music that is shorter than the piece's specified time signature. 3

pitch class A number between 0 and 11 (inclusive) that indicates the note name of any given note. For example, any C is of pitch class 0, any C♯/D♭ is of pitch class 1, any D is of pitch class 2, and so on up to B which is of pitch class 11. 29

relative major A major key relative to a given minor key such that the major key has the same key signature as the minor key. The relative major of a minor key is always the third degree of the minor scale. 25

relative minor A minor key relative to a given major key such that the minor key has the same key signature as the major key. The relative minor of a major key is always the sixth degree of the major scale. 25

repeat sign A section of notated music may be surrounded in repeat signs to indicate that the section should be played twice (or more indicated). 3

semi-tone The smallest unit of pitch in conventional Western music theory. Two adjacent notes on a piano, for example C and C♯ are said to be one semi-tone apart. 26, 40

time signature Specifies the beat value of a piece of music, and indicates the duration of a measure in terms of the number of beats. A time signature is composed of two numbers: a denominator and a numerator. The denominator indicates the value of a beat with respect to a whole note. For example, a denominator of 4 means the beat value is a quarter note. In conventional Western music the denominator is always a power of 2. The numerator specifies the number of beats per measure. For example, a time signature of $\frac{2}{4}$ means each measure has a duration of 2 quarter notes. 3

trill A quick alternation between two adjacent notes. 50

voice A single strand of melody within a piece of music. In the context of MICA, a voice is a single sequence of rhythmic durations with note values or rests assigned to each rhythmic duration. A single rhythmic duration within a voice may have more than one note (a chord) assigned to it. 3

Bibliography

Collins, N. (2009). Musical form and algorithmic composition. *Contemporary Music Review*, 28(1):103–114.

Cope, D. (1992). Computer modeling of musical intelligence in emi. *Computer Music Journal*, 16(2):69–83.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996.

Hedges, S. A. (1978). Dice music in the eighteenth century. *Music & Letters*, 59(2):180–187.

Hiller Jr, L. A. and Isaacson, L. M. (1957). Musical composition with a high speed digital computer. In *Audio Engineering Society Convention 9*. Audio Engineering Society.

Jones, K. (1981). Compositional applications of stochastic processes. *Computer Music Journal*, 5(2):45–61.

Kumar, R. and Jyotishree (2012). Blending roulette wheel selection & rank selection in genetic algorithms. *International Journal of Machine Learning and Computing*, 2(4):365–370.

Kuuskankare, M. and Laurson, M. (2006). Expressive notation package. *Computer Music Journal*, 30(4):67–79.

Laurson, M. and Kuuskankare, M. (2003). From rtm-notation to enp-score-notation. In *Journées d'Informatique Musicale.*

Laurson, M., Kuuskankare, M., Norlio, V., and Sprotte, K. (2013). Pwgl - a visual programming language. `http://www2.siba.fi/PWGL/`. Accessed: 2013-11-04.

Manaris, B., Romero, J., Machado, P., Krehbiel, D., Hirzel, T., Pharr, W., and Davis, R. B. (2005). Zipf's law, music classification, and aesthetics. *Computer Music Journal*, 29(1):55–69.

McAlpine, K., Miranda, E., and Hoggar, S. (1999). Making music with algorithms: A case-study system. *Computer Music Journal*, 23(2):19–30.

Miller, B. L. and Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131.

Moroni, A., Manzolli, J., Von Zuben, F., and Gudwin, R. (2000). Vox populi: An interactive evolutionary system for algorithmic music composition. *Leonardo Music Journal*, 10:49–54.

Nierhaus, G. (2009). *Algorithmic composition: paradigms of automated music generation.* Springer.

Privault, N. (2013). *Understanding Markov Chains: Examples and Applications.* Springer Singapore.

Purwins, H. (2005). *Profiles of Pitch Classes-Circularity of Relative Pitch and Key: Experiments, Models, Music Analysis, and Perspectives.* PhD thesis, Universitäts-bibliothek.

Sandred, O., Laurson, M., and Kuuskankare, M. (2009). Revisiting the illiac suite - a rule-based approach to stochastic processes. *Sonic Ideas/Ideas Sonicas*, 2:42–46.

Schwanauer, S. M. and Levitt, D. A. (1993). *Machine models of music.* MIT Press.

Srinivas, M. and Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(4):656–667.

Thierens, D. and Goldberg, D. (1994). Convergence models of genetic algorithm selection schemes. In *Parallel problem solving from nature-PPSN III*, pages 119–129. Springer.

Tokui, N. and Iba, H. (2000). Music composition with interactive evolutionary computation. In *Proceedings of the 3rd international conference on generative art*, volume 17, pages 215–226.

Vernon, P. (1942). The individuality of keys. *The Musical Times*, 83(1190):105–107.

Zipf, G. K. (1949). *Human behavior and the principle of least effort.* Addison-Wesley Press.