2012-11-26

Algorithmically Flexible Style Composition Through Multi-Objective Fitness Functions

Skyler James Murray
Brigham Young University - Provo

Algorithmically Flexible Style Composition Through Multi-Objective Fitness Functions

Skyler Murray

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dan Ventura, Chair
Neil Thornock
Sean Warnick

Department of Computer Science

Brigham Young University

November 2012

ABSTRACT

Algorithmically Flexible Style Composition Through Multi-Objective Fitness Functions

Skyler Murray
Department of Computer Science, BYU
Master of Science

Creating a fitness function for music is largely subjective and dependent on a programmer's personal tastes or goals. Previous attempts to create musical fitness functions for use in genetic algorithms lack scope or are prejudiced to a certain genre of music. They also suffer the limitation of producing music only in the strict style determined by the programmer. We show in this thesis that *musical feature extractors* that avoid the challenges of qualitative judgment enable creation of a multi-objective function for direct music production. Multi-objective fitness functions enable creation of music with varying identifiable styles. With this system we produced three distinct groups of music which computationally cluster into distinct styles as described by the set of feature extractors. We also show that knowledgeable individuals make similar clusters while a random sample of people make some similar and some different clusterings.

Keywords: Music Composition, Genetic Algorithms, Feature Extractors

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computational music composition is a challenging area of computational creativity and numerous studies attempt a variety of approaches to produce music with computers. However, music theory contains many rules and conventions that are difficult to formalize and the intricacies at many levels, from local to global, create a complexity that makes the production of convincing music difficult.

Despite the difficulties, many computational music systems exist that challenge the perceived limitations of computers. One very successful example is Cope's Experiments in Musical Intelligence (EMI) system which can mimic the compositional style of history's greatest composers. His system is so effective that the output is indistinguishable from the source composers' own compositions [6]. Other examples include Anders and Miranda [1] demonstrating an effective method for producing chord progressions that follow established rules and Tanaka et al. [18] encoding the rigorous rules of two-part counterpoint into stochastic models that produces convincing counterpoint, demonstrating the breadth of solutions that exist for computational music composition.

While many successful approaches exist, Genetic Algorithms (GA) offer the greatest flexibility for producing varied musical outputs. The many ways that a GA's fitness function and genomes can be designed allow this versatility. Freitas and Guimaraes [9] show how genetic algorithms can achieve this. Their use of multiple fitness functions to harmonize melodies leads to two classes of outputs determined by which fitness function they weight higher. This leads to convincing harmonization that utilized one of two different styles—

simplicity or dissonance. Their work demonstrates the power genetic algorithms have to produce a variety of styles when driven by multiple fitness functions.

Genetic algorithms can be successfully applied to the musical domain, but implementing an effective and useful fitness function remains a challenge [5]. The problem lies in quantifying how *good* a piece of music is. This is largely subjective and remains a major hurdle for the domain. Currently the two most common approaches are human-in-the-loop and algorithmic.

Interactive Genetic Algorithms (IGA) is an approach that involves human input as the fitness function but suffers from throughput issues—the fitness bottleneck [2]. Music is best experienced one piece at a time while listening from beginning to end. The time involved in the process makes rating larger populations through a human evaluator impractical. Biles' work [2] on an evolutionary composition based system—GenJam—produces improvisational jazz lines through an evolutionary based system that uses input from a human rater and he agrees that this approach leads to low throughput from his system. Biles attempts to implement a neural network [4] to overcome this challenge but is unable to produce the same quality of results achieved by the human rater. Biles' efforts to move away from IGA[3] show that another approach is desirable.

Implementing an automated fitness function allows for quicker processing but limited evaluation of the musical output due to the narrow scope of most implemented fitness functions. Whether the fitness function is designed to look for specific 4-part harmony rules [13] or members of the diatonic scale, the function limits the output's scope. Because both human-in-the-loop and programmed fitness functions have significant drawbacks, a new method is needed—an approach that avoids the fitness bottleneck and allows for a more flexible way to escape the programmer's bias.

We present a system that addresses these challenges and allows a genetic algorithm to flexibly produce multiple unique styles. We present an array of feature extractors as a solution. The feature extractors do not place a fitness score on a piece of music; they analyze

the musical output to determine where in the musical space the output lies. Individual extractors analyze separately the harmony, the distribution of rhythms, the overall shape of the lines, self-similarity, repetition and other aspects of the output. A multi-objective fitness function then can use a weighted combination of feature extractors targeted to specific scores to produce a fitness which the GA uses to drive the evolutionary model. This system avoids the challenges inherent in creating musical fitness functions and produces a distinct musical style determined by the fitness function weights and feature targets.

To evaluate our multi-objective fitness function approach to producing a particular style we showed outputs from the system to a number of Brigham Young University School of Music faculty who analyzed them for stylistic similarities and clustered the outputs. We did a study as well with human raters to construct a distance matrix for the produced music and clustered according to that matrix. We also clustered the results based on the outputs of the feature extractors as a means of secondary validation. These three qualitatively different evaluation methods all confirm that the system can produce distinct and recognizable styles of music.

Chapter 2

Related Work

Genetic algorithms are an often used approach to computational music composition. We consider here the numerous examples that exist of genetic approaches to music production as well as specific implementations of fitness functions and how to augment their effectiveness in this domain by use of feature extractors.

2.1 Genetic Algorithms

Genetic Algorithms (GA) [10] [11] are an evolutionary method of optimization. GAs offer a way to solve complex problems without a specifically tailored search algorithm and a way to overcome the shortcomings of many other often-used optimization algorithms [7].

GAs find a solution through optimization of a fitness function using a *population* of possible solutions—*individuals*. The *individuals* are randomly initialized as binary or real-valued strings that represent an *individual's* genome. Their fitness is measured by the GA based on certain criteria. The GA chooses *individuals* who will populate a *mating*

Algorithm 1 Genetic Algorithm

```
Initialize population
while not done do
    Calculate fitness for all individuals
    Order individuals by fitness
    Create pobabilistic mating pool of individuals
    Create new offspring from mating pool using crossover operators
    Use mutation operator on new offspring
    Select subset of offspring and current population as new population
Ouput m individuals with fitness > T
```

pool. Offspring are produced during the *reproduction* phase through a *crossover* operation. *Mutation*—random alterations to an *individual's* genome—is also possible as part of the reproduction phase to ensure complete coverage of a search space. The GA terminates when it reaches a predetermined criterion—often when the population reaches a high enough mean fitness score [5]. See Algorithm 1.

The wide applicability of GAs [7] shows promise for applications in music generation. Phon-Amnuaisuk, Tuson, and Wiggins [17] give an overview of many issues to consider when combining GAs and music, show several examples of successfully using GAs to harmonize pieces of music, and show the necessity of encoding a great deal of musical knowledge and practice in the GA operators. Biles calls these *musically meaningful mutations* [2]. Without this knowledge it is difficult to produce meaningful music.

It is common practice to encode these *musically meaningful mutations* into any attempt at producing music with a GA. The standard implementation of GAs employs a binary representation for genomes and mutations are often random flipping and swapping of bits. This is often not conducive to applying a GA to music. Freitas and Guimarães [9] show the importance of using *musically meaningful mutations*. Besides implementing musical versions of crossover and mutation, they use methods that swap notes between measures, randomize chords, and copy other measures. Their melody harmonization system creates near human quality harmonizations, showing how successful GA operators can be when empowered with specific musical knowledge. Encoding this type of knowledge is an essential part of our GA implementation.

2.2 Fitness Functions

At the core of any GA is a fitness function. The fitness function drives the evolutionary process of a GA by assigning a fitness score to members of the population. In most GAs this score is used to determine which members will survive to the next iteration and produce

offspring. Without an accurate and useful fitness function the GA will never converge to a meaningful solution.

Building an effective fitness function for music offers unique challenges. Music is often a nebulous concept that is hard to place value on and to precisely evaluate. It is no easy task and music critics make a living critiquing performances and compositions. Critics often use subjective terminology suggesting how music affects the emotions. Given that computers do not have emotions, or much experience with music, it becomes a difficult task for a computer to reliably evaluate a piece of music.

When applied to a limited scope of musical attributes a fitness function can effectively drive a population to converge to high fitness scores. Freitas and Guimarães [9] use a two fitness-function approach that scores harmonization outputs from their system. One function scores the outputs based on their simplicity and adherence to common harmonization rules, the other based on the level of dissonance. After many generations of harmonization, the surviving individuals with high fitness scores in either category exhibit the traits which score highly in the respective fitness functions. From this they produce two different types of results. Their work shows how two differing fitness functions enable the generation of two styles of outputs.

2.3 Feature Extractors

Feature extraction from music offers a variety of applications in computer science. Yip et. al [19] show how extracting features from music is useful in cataloging melodies. McKay [14] creates a successful music genre classification system based on music feature extraction and later published an open-source library [15] of the same software. Extraction of musical features also holds potential for enabling better music generation. Musical features offer the potential for representation of different scales. Possible features include the distribution of notes, rhythms, harmonies, extrema, and how they interrelate. In particular, we believe feature extractors are an important part of building a more intelligent fitness function.

Chapter 3

System Design

This thesis presents a new adaptation of previous approaches to music generation. This adaptation addresses the fitness bottleneck and the inflexibility of rigidly designed fitness functions. Specifically this approach:

- Uses an array of feature extractors which feed into a multi-objective fitness function
- Can target any part of the spectrum of features with a targeting function
- Drives the evolutionary process with a multi-objective fitness function weighted by the input of the feature extractors
- Produces music of varying, yet identifiable, styles

3.1 Genetic Algorithms

Our approach to GAs modifies several aspects of the commonly used GA [10] [11] while keeping the overall algorithm intact (see Algorithm 1). As opposed to traditional GAs where *individuals* are represented by a binary string, our implementation represents *individuals* with a string of note names. Larger departures from the traditional GA implementation will be in the crossover and mutation operators, for which we will implement *musically meaningful operators*. The fitness function implementation will also be a departure from common approaches to music. These differences are discussed in more detail below.

3.2 Musically Meaningful Operators

Many implementations exist in the literature for *musically meaningful operators* [12] [17] [9] [13] [16]. Many of these approach the problem by implementing more than just the traditional crossover and mutation operators. We believe that implementing just the traditional crossover and mutation operators in a musically meaningful way enables production of the desired music. These standard operators are what we implemented.

We implemented a standard one-point crossover where the splitting point in two individuals is randomly chosen between two notes. Mutation is typically a random bit flip in standard implementations but this is not suitable for our purposes. We implemented mutation as an alteration to one note in the string of notes by probabilistically altering its pitch up or down. The degree of change is chosen from a standard distribution of note values with $\sigma = 2.0$ and the result is quantized to integer values. These changes to the standard GA operators allow for the mutation and crossover phases of our GA to happen without significant computational overhead. Selection for breeding is done tournament style [10] with a selection pressure of 0.5.

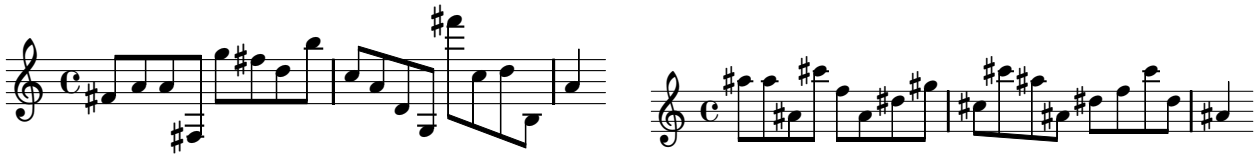
3.3 Feature Extractors

A set of feature extractors provides the inputs to the fitness function to give each individual a fitness score. Each feature extractor analyzes an individual I and computes a function

$$e : \mathbb{I} \rightarrow [0, 1]$$

where $\mathbb{I}$ is the set of all individuals (sequences of musical pitches).

The function output reflects how well that particular feature is represented in the individual. As an example, one feature is based on what percent of the notes fit into the musical key of G major. 100 percent of the notes falling in the key of G major would lead to a score of 1, but if 0 of them fall into the key then the score would be 0—See Figure 3.1.



(a) G Major Feature Score of 1 (b) G Major Feature Score of 0

Figure 3.1: G Major Feature Examples

Similar features exist for the spectrum of key signatures, allowing the system to determine the dominant key or keys of a piece depending on how the features score.

We introduce the following notation that we will use in describing the feature extractors. An individual $I \in \mathbb{I}$ is a sequence of notes; $I = i_1 i_2 i_3 ... i_n$. An individual note i can take any pitch value in a four-octave range, with each value represented as a number in the interval $[0, 48]$, so $i_j \in [0, 48], 1 \leq j \leq n$.

The feature extractors are implemented as follows:

- **Self-Similarity:** Measures how often repeating interval sequences occur in I and uses this as a measure of self-similarity—if the same interval sequences occur often, the piece is more self-similar than if many different interval sequences occur less frequently.

$$SelfSimilarity(I) = \max\left\{1, \frac{2\mu}{|I|}\right\}$$

where

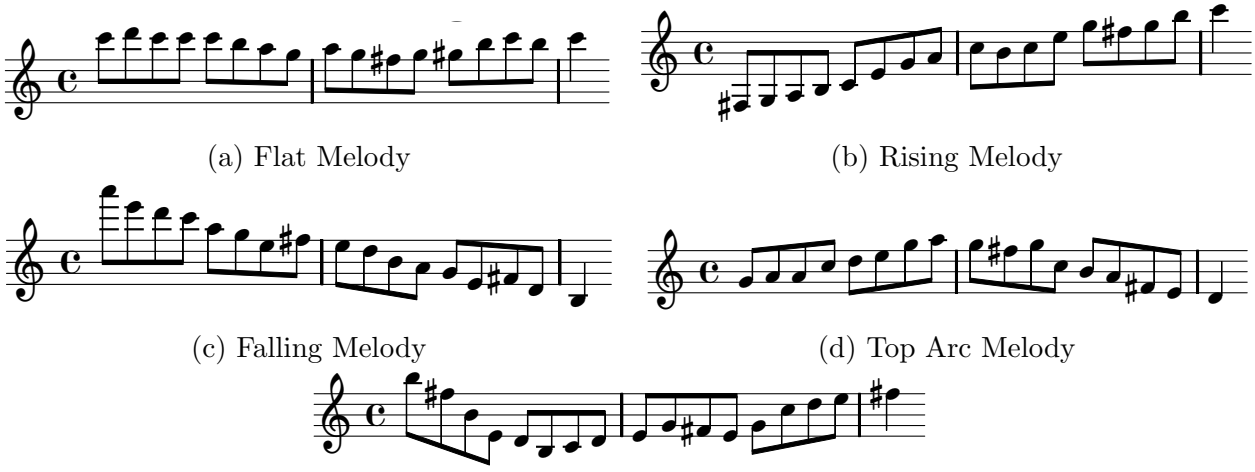
$$\mu = \frac{1}{|S|}\sum_{s \in S} count_s(I)$$

and S is the set of all interval sequences of length 2 that appear in I, and $count_s(I)$ is the number of times interval sequence s occurs in I.

- **Melody Shape:** A set of five functions that calculate how well I fits a particular melody shape. The five melody shape functions are: $FlatMelody(I)$, $RisingMelody(I)$, $FallingMelody(I)$, $TopArcMelody(I)$, $BottomArcMelody(I)$. These shapes are illustrated in Figure 3.2. A linear regression is used to calculate slope

m and mean square error ϵ. We calculate $m_{max} = \frac{NoteRange}{|I|}, NoteRange = 49$. The top arc and bottom arc shapes are calculated by splitting the melody in half and calculating a rising and falling melody score on the first and second half respectively. For the bottom arc shape the reverse is done. To avoid discontinuities, an overlap of two notes is used in splitting the melody into two parts.

$$MelodyShape(I) = \left(1 - \frac{m_{max} - m}{2m_{max}}\right)\left(1 - \frac{\epsilon^2}{\epsilon^2 + 10000}\right)$$



(a) Flat Melody

(b) Rising Melody

(c) Falling Melody

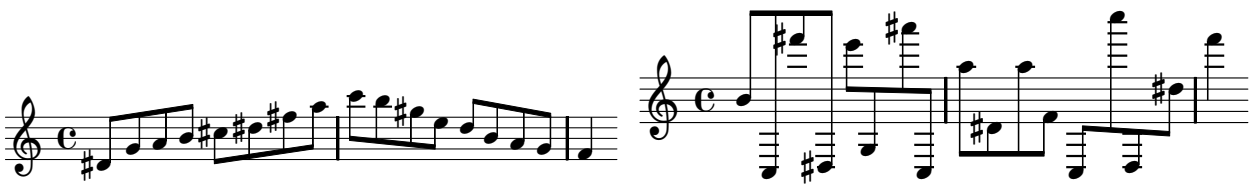
(d) Top Arc Melody

(e) Bottom Arc Melody

Figure 3.2: Melody Shapes

- **Linearity:** Measures how angular the notes in I are. For an example see Figure 3.3. Approximates the second derivative at each note in I using the absolute values of the notes to compute the approximation. α is a smoothing term to adjust how quickly the linearity approaches 1. $S(I)$ is an approximation for the second partial derivative, similar to a Laplacian kernel. $\beta = -1, \kappa = 2, \alpha = 15$.

$$Linearity(I) = \frac{S(I)^2}{S(I)^2 + \alpha},\ S(I) = \sum_{k=2}^{n-1} |\beta i_{k-1} + \kappa i_k + \beta i_{k+1}|$$



(a) Linearity score close to 1 (b) Linearity score close to 0

Figure 3.3: Linearity feature examples

- **Key Prevalence:** 12 functions for each possible key center. Measures the proportion of notes from I that represent that key. Here, $1 \leq j \leq 12$, and Key_1 is C Major, Key_2 is G Major...Key_{12} is F Major.

$$KeyPrevalence_j(I) = \frac{|K_j|}{|I|},\ K_j = \{i \in I | i \in Key_j\}$$

- **Tonality:** Uses the output of Key Prevalence functions. If all key centers are equally dominant then output a 0 (atonal), but if a key center is completely dominant then output a 1. Here, $1 \leq j \leq 12$.

$$Tonality(I) = \max_j\ KeyPrevalence_j(I) - \frac{1 - \max_j\ KeyPrevalence_j(I)}{n-1}$$

- **Distribution of Pitch:** We denote $|i|_p$ to mean the *pitch class* of a note—i.e. a C is in the same pitch class no matter which octave it is in. $|i|_p$ takes values in the interval $[1, 12]$. When all 12 pitch classes are used equally output 1 but when only a single pitch class is used output 0. Here, $1 \leq j \leq 12$.

$$PitchDistribution(I) = \frac{\frac{n - \max_j\ P_j(I)}{11}}{n} \cdot 12,\ P_j(I) = \sum_{k=1}^{n} \delta(j, |i_k|_p)$$

- **Range of Pitch:** Scores how much of the full range of pitches are utilized by I. A score of 0 implies none of the range used while a score of 1 means the whole range is

used. $P(I)$ calculates a weighted percentage of pitches in the four-octave range covered by I. We use a non-linear scaling with $\gamma = 15$ that will weight the use of the first two octaves more importantly than notes in the third and fourth.

$$PitchRange(I) = \frac{\gamma P(I)^2}{\gamma P(I)^2 + 1}$$

- **Ascending/Descending Interval Prevalence:** Similar to $KeyPrevalence()$ feature. Intervals over an octave in size are reduced to "pitch class intervals" (their between octave equivalent). Separate functions are used for ascending and descending intervals. This leads to 24 separate functions. Here, $0 \leq j \leq 11$.

$$AscendingIntervalClassPrevalence_j(I) = \frac{\sum_{k=1}^{n-1} \delta(j, (i_{k+1} - i_k) \bmod 12)}{n-1}$$

$$DescendingIntervalClassPrevalence_j(I) = \frac{\sum_{k=1}^{n-1} \delta(-j, (i_{k+1} - i_k) \bmod 12)}{n-1}$$

- **Interval Class Prevalence:** Similar to $KeyPrevalence()$ feature. Ascending and descending intervals return the same value and intervals over an octave in size are reduced to their between-octave equivalent. This leads to 12 separate functions. Here, $0 \leq j \leq 11$.

$$IntervalClassPrevalence_j(I) = \frac{\sum_{k=1}^{n-1} \delta(j, |i_{k+1} - i_k| \bmod 12)}{n-1}$$

- **Inverted Interval Prevalence:** This metric stems from music theory in which the inversion of an interval is treated as the same class of interval. A major 2nd inverted is a minor 7th, minor 2nd inverted is a major 7th, and so on. This leads to seven interval classes; Unison/Octave, m2/M7, M2/m7, m3/M6, M3/m6, P4/P5, Tritone

(m = minor, M = Major, P = perfect). Here, $0 \leq j \leq 7$.

$$InvertedIntervalPrevalence_j(I) = \frac{\sum_{k=1}^{n-1} \delta(j \text{ or } 12-j, |i_{k+1} - i_k| \bmod 12)}{n-1}$$

- **Over the Octave Interval Prevalence:** Calculates the percentage of intervals that are greater than an octave in size.

$$OverOctaveIntervalPrevalence_j(I) = \frac{\sum_{k=1}^{n-1} T(|i_{k+1} - i_k|)}{n-1}, T(n) = \begin{cases} 1, & \text{if } n > 12 \\ 0, & \text{if } n \leq 12 \end{cases}$$

- **Intervalic Distribution:** Only one interval used, whether ascending or descending, output a 0. If all 12 intervals are equally used then output a 1.

$$IntervalDistribution(I) = \frac{1 - \max\limits_{j} \ IntervalClassPrevalence_j(I)}{11} \cdot 12, \ 0 \leq j \leq 11$$

3.4 Target Function

While the set of feature extractors adds diversity to what the GA produces, we still need a way to target any specific feature score. Without a way to target a particular feature score, the GA would only converge to feature values close to 1. To overcome this, we use a secondary function as a target function where the score is related to how closely the feature value comes to a target value t. The function takes the score to target t, which is in the range $[0, 1]$, and the output of the feature extractor $e(I)$ as input and computes the score as follows.

$$FeatureScore(t, e(I)) = \frac{-1}{(x(t) - t)^2} (e(I) - t)^2 + 1, \ x(t) = \begin{cases} 1, & \text{if } t < 0.5 \\ 0, & \text{if } t \geq 0.5 \end{cases}$$

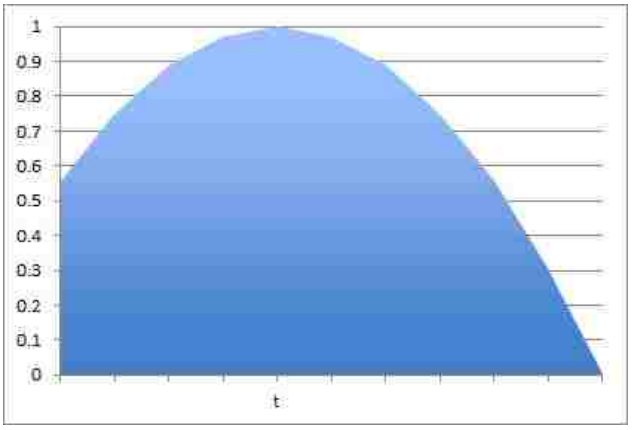


Figure 3.4: Target Function Example: The feature score is highest at t and slopes off in both directions

In this way the GA can target any range of values from the feature extractor by adjusting the value for t and using the output in the fitness function. Figure 3.4 represents how this works.

3.5 Multi-Objective Fitness Function

The multi-objective fitness function is the major contribution of this thesis, providing a flexible framework that produces a variety of musical styles. By using a linear combination of weighted outputs from the feature extractors, the multi-objective fitness function biases the musical outcome, with the weights acting as "preferences." Thus,

$$f(I) = \sum_{e \in E} \alpha_e FeatureScore(t_e, e(I))$$

is a multi-objective fitness function that represents a stylistic musical preference, parameterized by the set of targets $\{t_e\}$ the set of weights $\{\alpha_e\}$. Here, E is the set of feature extractors, the t_e are the target feature values and the α_e weight the extractors, with each different setting of the weights/targets corresponding to some different musical style. Note that there is some interplay between the two sets of parameters but that they serve different

functions. The targets control the *quality* of different musical features, while the weights control their *importance*.

For example, consider the following function.

$$\begin{aligned} f(I) &= \frac{2}{3} FeatureScore(0.9, KeyPrevalence_2(I)) \\ &\quad + \frac{1}{3} FeatureScore(0.5, PitchRange(I)) \end{aligned}$$

This function scores most highly music that makes heavy use of the key of G major and employs a moderate range of pitches. The key feature is twice as important as the pitch range feature and no other features are considered at all. Of, course, non-linear combinations and negative weighting of features are also also potential representations for musical styles; however, here we will limit ourselves to the linear, positive weight case.

The power of this system lies in the variety of feature extractors—detailed above—and the ability to combine them in arbitrary ways. With this flexible approach, our system has the ability to produce a variety of styles depending on how features are targeted and weighted. A challenge of this approach is that the number of feature extractors creates a complexity that can result in slow convergence times. This may be ameliorated, to some extent, by placing practical bounds on the fitness functions. For example, we can limit the number of *KeyPrevalence*() extractors that can receive non-zero weightings.

Our system built with these attributes offers a flexible platform for music generation and produces a variety of identifiable styles of music. While our system cannot create such popular styles as hip-hop and country, the variety of styles are distinguishable from one another and self-similar within the same style.

Chapter 4

Experimental Method

Here we present our method for validating our approach to computer music production. We detail the production of music as well as our approach to validating our results.

4.1 Cluster Production

We produced three clusters of melodies for our validation methods. The clusters contained 10 melodies each. Clusters were based on three different sets of the production parameters we chose. While we had a diverse set of feature extractors to use, we settled on using a set of seven features for producing the three clusters. These features, with associated weights and targets, are outlined in Table 4.1.

An example of each style is given in Figure 4.1. We give every example from each produced cluster in Appendix A.

Cluster A features a majority of notes in the key of G Major, uses a wide range of pitches, conforms mostly to a top arc shaped melody, has a linearity score in the middle

	Cluster A		Cluster B		Cluster C	
	Weight	Target	Weight	Target	Weight	Target
$SelfSimilarity$	20	0.05	20	0.05	20	0.2
$TopArcMelody$	20	0.8	20	0.6	20	0.8
$IntervalClass_0$	10	0.05	10	0.05	10	0.0
$IntervalClass_5$	10	0.04	10	0.04	10	0.4
$PitchRange$	10	0.9	10	0.9	10	0.4
$KeyPrevalence_2$	30	1.0	30	1.0	30	0.4
$Linearity$	20	0.5	20	0.8	20	0.8

Table 4.1: Cluster features, weights and targets

of the range so it presents more and larger jumps in opposite directions. These traits are apparent in Figure 4.1a.

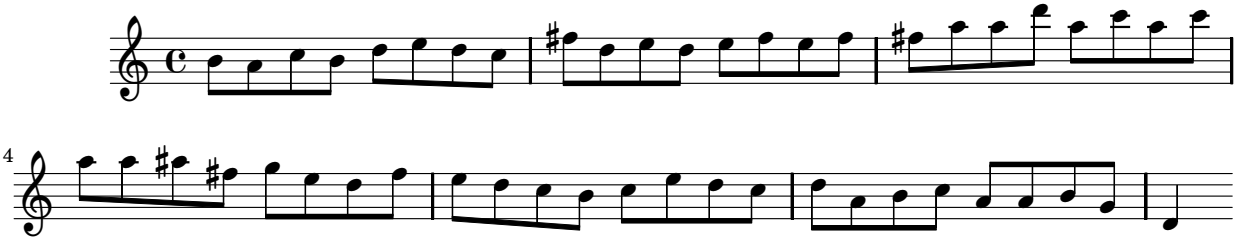
Cluster B—Figure 4.1b—shows many of the same features as Cluster A with a few differences. It conforms less well to the top arc shape and with a higher linearity target it features smoother lines with less large jumps in opposite directions and more consistent use of the same interval.

Cluster C—Figure 4.1c—differs from the other clusters in many noticeable ways. The raised target for self similarity produces music with more frequent and common interval patterns. The changes to the $IntervalClass_0$ and $IntervalClass_5$ targets produce melodies with fewer repeated notes and a predominant use of the Major 4th Interval. It also uses a smaller range of notes, produced by the reduction in the target score for pitch range and dropping the target for key prevalence to 0.4 drives more than half of the notes to be outside of G Major.

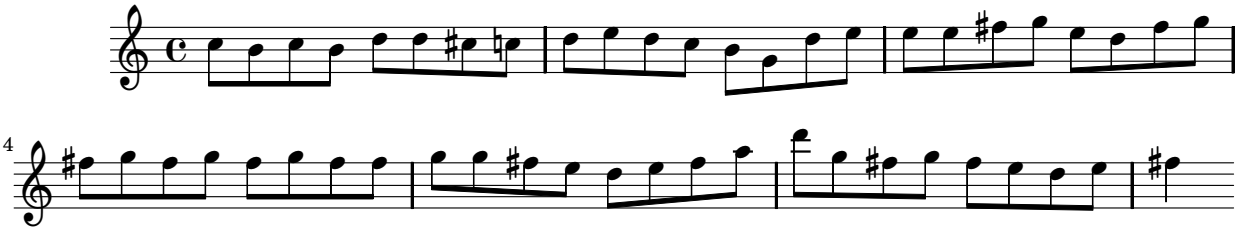
We produced 10 different melodies for each of the styles (referred to here as A0,...,A9, B0,...,B9, C0,...,C9) using Algorithm 2. This is done for each of the three clusters with the feature weights and targets in Table 4.1 used for each cluster. We initialize the GA with a population of 40 melodies with random pitches in the range of four octaves.

Algorithm 2 Generator for producing melodies

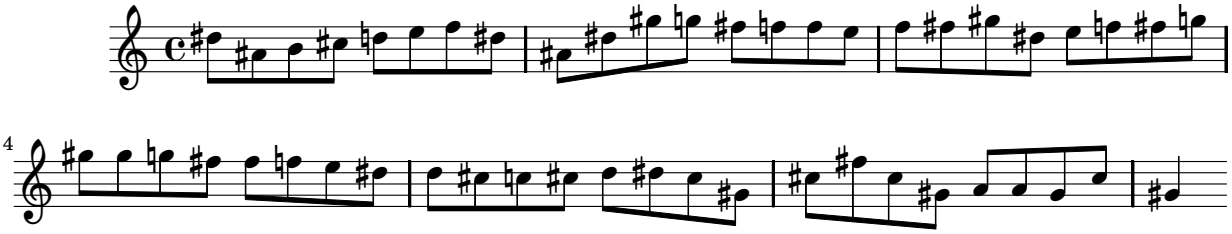
```
C = {}
while |C| < 10 do
  T = 0.99, bestfitness = 0.0, count = 0
  while bestfitness < T do
    bestfitness=GA(f)
    if bestfitness is not improving then
      restart GA
      count = count + 1
      if if count = 3 then
        T = T − 0.01
        count = 0
  C=C ∪ individual with bestfitness > T
return C
```



(a) Cluster A Example



(b) Cluster B Example



(c) Cluster C Example

Figure 4.1: Cluster Examples

4.2 Clustering Experiments

To show that our system produces music of multiple identifiable styles we employed three different clustering techniques.

- Agglomerative clustering based on the vector of features from the individual outputs
- Agglomerative clustering using a human produced distance metric
- Clustering by knowledgeable music faculty

These approaches are detailed in the following sections.

4.2.1 Agglomerative Feature Clustering

This first approach utilizes a number of different distance metrics and clustering algorithms to cluster the outputs from our system. With the clusters produced above, a distance matrix was computed utilizing the raw scores of the selected features as a position vector. Using this position vector, we produced a distance matrix with each of the following five distance metrics.

- Euclidean Distance
- Euclidean Squared Distance
- Manhattan Distance
- Maximum Distance
- Cosine Similarity

Using *MultiDendrograms* [8], we clustered the five distance matrices with five different agglomerative clustering algorithms:

- Single Linkage
- Complete Linkage

- Unweighted Average
- Weighted Average
- Joint Between-Within

This produced 25 different dendrograms. To evaluate the accuracy of our clustering we computed the following four metrics on each clustering.

- Purity
- Rand Index
- F-Measure
- Normalized mutual information

4.2.2 Human Distance Metric

Our second approach to clustering utilized the same three clusters of music produced above. We utilized a random sample of people to create a distance matrix for the produced music clusters. The goal was to see if a random sample of people will create a similar clustering as the feature clustering.

As the distance matrix compares every music example to itself and to every other example, for 30 music examples the distance matrix is 30x30. We assume that the reflexive comparisons are a distance of 0 which leaves 870 comparisons. We also assume symmetry, cutting the number of comparisons needed to 435.

In our study we used 22 people to each reproduce 20 comparisons or distances (the 22nd participant only did 15). For each of the 20 comparisons in the study they were asked to listen to two music examples and rate how similar or dissimilar the two examples are on a scale of 0 to 10. They were also asked to describe why they rated the examples so similar or dissimilar.

From the participants' responses we recreated the distance matrix and created dendrograms using the five clustering algorithms listed in the previous section. We then also computed the four clustering metrics listed in the previous section.

4.2.3 Knowledgeable Experts Clustering

While music experience can vary greatly in a random sample of people, utilizing faculty from the School of Music provides some measure of musical "expert knowledge". They routinely analyze music, offering a way to see how people with significant musical experience cluster the outputs from our system. We used three members of the Brigham Young University School of Music faculty for our study.

We chose a random subset of the 30 melodies: five of stye A, $\{\text{A0}, \text{A2}, \text{A6}, \text{A7}, \text{A9}\}$; four of style B, $\{\text{B3}, \text{B4}, \text{B6}, \text{B9}\}$; and three of style C, $\{\text{C0}, \text{C2}, \text{C7}\}$. We gave each faculty member recordings and the musical scores for each example. We asked them to do the following:

- Listen to and study the scores for each selection.
- Analyze and list distinctive traits for each selection.
- Group the selections into a number of different groups based on common traits that you identify while ignoring the similarity of rhythm.
- List the traits that differentiate each group from the other.

From the faculty clustering we computed the same four clustering metrics used in the previous two sections.

Chapter 5

Results

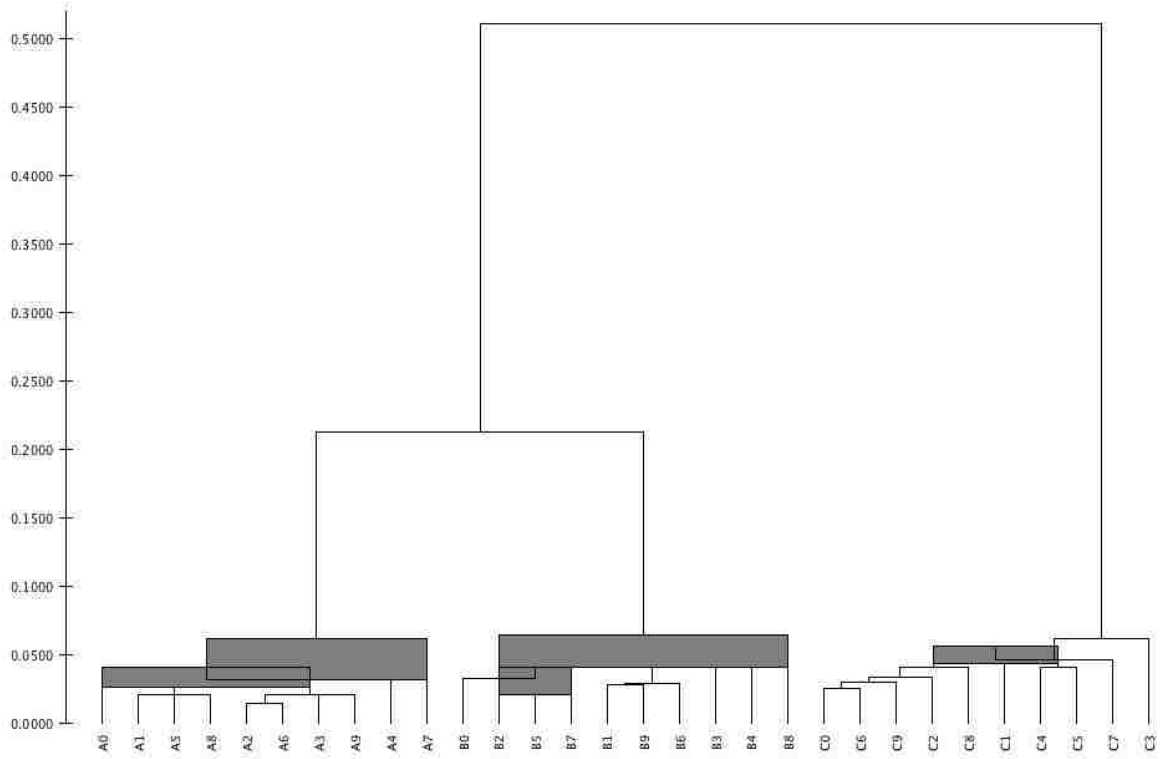
Here we present the results from the three studies we performed.

5.1 Agglomerative Feature Clustering

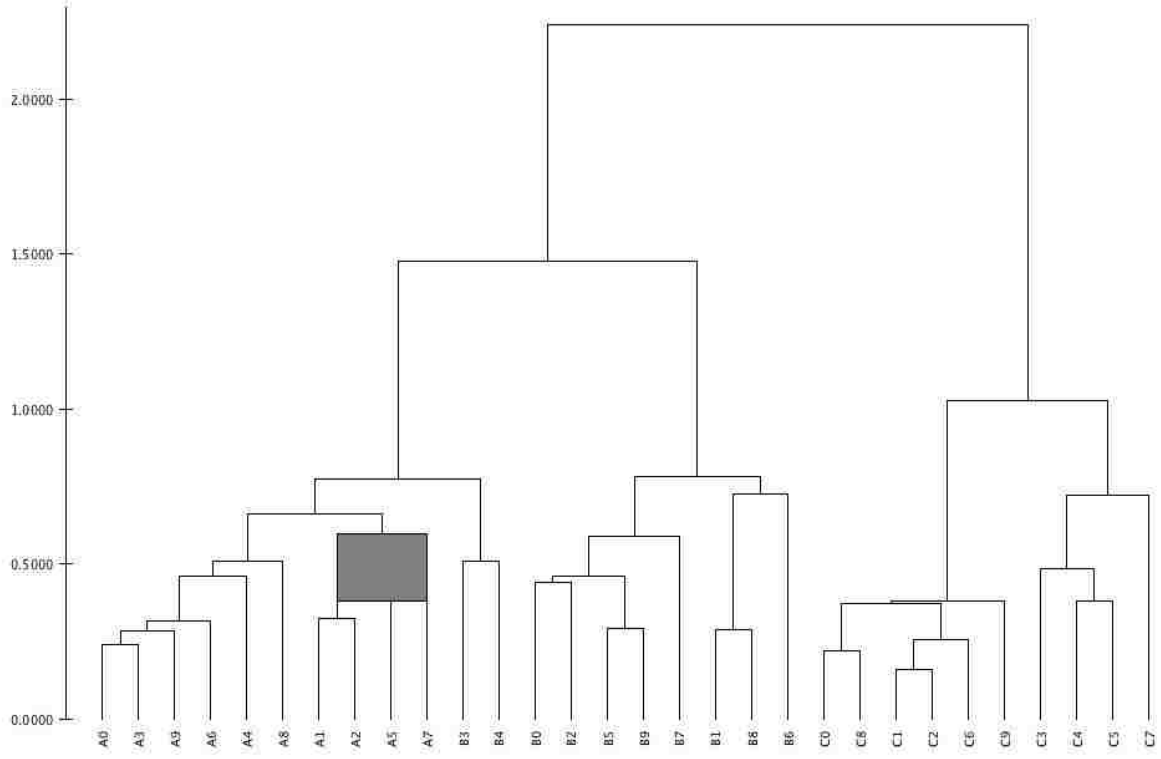
Using the clusters produced above we calculated a distance matrix based on five different distance metrics: Euclidean, Euclidean Squared, Manhattan, Maximum, and Cosine-Similarity. Then using *MultiDendrograms* on each distance metric we produced dendrograms based on five different clustering algorithms: Single Linkage, Complete Linkage, Unweighted Average, Weighted Average, and Joint Between-Within. With these 25 dendrograms we calculated Purity, NMI, F-Measure, and Rand Index (RI) for each clustering.

Nineteen of the 25 clusterings correctly classified all but two of the melodies. These two melodies from the second cluster were classified as coming from the first cluster. These results led to the majority of the scores reported in Tables 5.1, 5.2, 5.3, and 5.4. The remaining six clusterings correctly clustered all of the results. These were the Manhattan metric using the Joint Between-Within clustering algorithm and the Maximum distance metric using all five algorithms. Figures 5.1a and 5.1b show an example of the dendrograms with every individual correctly classified and with only two melodies incorrectly classified respectively. The complete set of dendrograms are found in Appendix B.

While these metrics show a very high quality of clustering, this is what we expected to happen. Given that the distance measures are computed using the attributes the system used for representation, it is not surprising that these clustering approach performed so well.



(a) Completely correct classification



(b) Two incorrect classifications

Figure 5.1: Dendrograms

	Single Linkage	Complete Linkage	Unweighted Average	Weighted Average	Joint Between-Within
Euclidean	0.93	0.93	0.93	0.93	0.93
Euclidean Squared	0.93	0.93	0.93	0.93	0.93
Manhattan	0.93	0.93	0.93	0.93	1.00
Maximum	1.00	1.00	1.00	1.00	1.00
Cosine-Similarity	0.93	0.93	0.93	0.93	0.93

Table 5.1: Purity

	Single Linkage	Complete Linkage	Unweighted Average	Weighted Average	Joint Between-Within
Euclidean	0.84	0.84	0.84	0.84	0.84
Euclidean Squared	0.84	0.84	0.84	0.84	0.84
Manhattan	0.84	0.84	0.84	0.84	1.00
Maximum	1.00	1.00	1.00	1.00	1.00
Cosine-Similarity	0.84	0.84	0.84	0.84	0.84

Table 5.2: NMI

	Single Linkage	Complete Linkage	Unweighted Average	Weighted Average	Joint Between-Within
Euclidean	0.87	0.87	0.87	0.87	0.87
Euclidean Squared	0.87	0.87	0.87	0.87	0.87
Manhattan	0.87	0.87	0.87	0.87	1.00
Maximum	1.00	1.00	1.00	1.00	1.00
Cosine-Similarity	0.87	0.87	0.87	0.87	0.87

Table 5.3: F-Measure

	Single Linkage	Complete Linkage	Unweighted Average	Weighted Average	Joint Between-Within
Euclidean	0.91	0.91	0.91	0.91	0.91
Euclidean Squared	0.91	0.91	0.91	0.91	0.91
Manhattan	0.91	0.91	0.91	0.91	1.00
Maximum	1.00	1.00	1.00	1.00	1.00
Cosine-Similarity	0.91	0.91	0.91	0.91	0.91

Table 5.4: RI

	μ_A	μ_B	μ_C	$\lvert\mu_A - \mu_B\rvert$	$\lvert\mu_A - \mu_C\rvert$	$\lvert\mu_B - \mu_C\rvert$
$SelfSimilarity$	0.0617	0.0684	0.1145	0.0067	0.0528	0.0461
$TopArcMelody$	0.6302	0.5160	0.5607	0.1142	0.0695	0.0447
$IntervalClass_0$	0.0837	0.1163	0.0714	0.0327	0.0122	0.0449
$IntervalClass_5$	0.0551	0.0388	0.2367	0.0163	0.1816	0.1980
$PitchRange$	0.7902	0.7070	0.4474	0.0831	0.3427	0.2596
$KeyPrevalence_2$	0.8531	0.7878	0.4571	0.0653	**0.3959**	**0.3306**
$Linearity$	0.4963	0.7550	0.7127	**0.2587**	0.2164	0.0423

Table 5.5: Average feature scores for each cluster and the between cluster differences for the average feature scores. The maximally dissimilar scores are in bold.

We also note the success of the maximum distance metric in correctly classifying every example. This metric bases its score entirely on the maximally dissimilar features of each cluster of music. Examining the feature scores for each cluster and individual reveals the differentiating feature. Clusters A and B differ most in the *Linearity* feature extractor while both clusters A and B differ most from cluster C in the $KeyPrevalence_2$ feature extractor. Table 5.5 summarizes this information. Examining the features and targets in Table 4.1 shows that the same holds true for what was targeted in the fitness function.

5.2 Human Distance Metric

Using the responses from the random sample of people we created a distance matrix. We used the same clustering algorithms as outlined above to produce dendrograms of the results. The same four clustering metrics were also used to qualify the results. These scores appear in Table 5.6. Figure 5.2 shows the clustering produced using the Weighted Average algorithm. The remaining dendrograms are found in Appendix B.

While the clusterings based on the random sample of people are not as precise as the feature based clusterings, several specific comparisons of musical example pairs show exactly what we hoped they would show. The scores they received were also about what we would expect them to receive. For example,

- "The second piece had much more dissonance and half-steps" (Comparing A7 to C3)

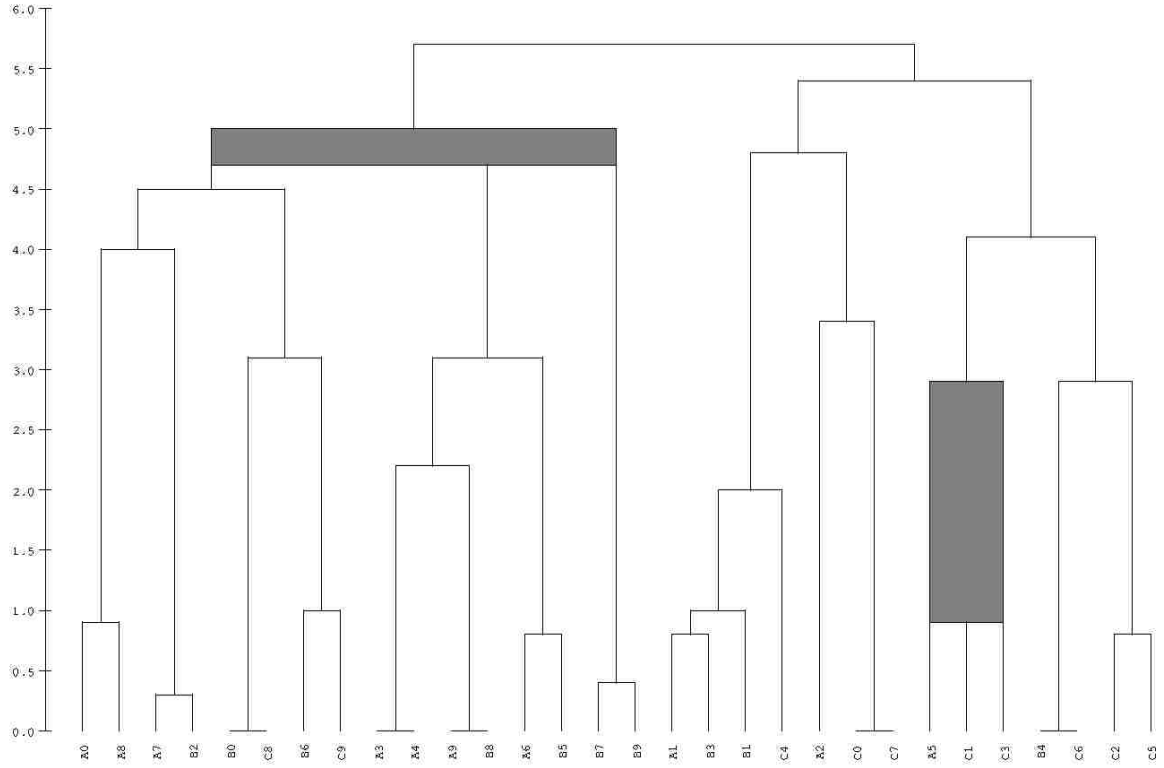


Figure 5.2: Dendrogram from Human Distance Metric

	Single Linkage	Complete Linkage	Unweighted Average	Weighted Average	Joint Between-Within
RI	0.36	0.65	0.56	0.61	0.58
F-Measure	0.45	0.33	0.45	0.43	0.35
NMI	0.15	0.23	0.18	0.13	0.34
Purity	0.43	0.60	0.53	0.50	0.53

Table 5.6: Random Sample Clustering Metrics

- "Although they had different ranges, both moved very chromatically" (Comparing C1 to C7)
- "seemed almost like different parts of the same song" (Comparing B7 to B9)

While there are a number of similar examples, there are also responses and scores that resulted from musical aspects that our system does not consider. These aspects led to scores different from the feature distance metrics. Examples include

- "In some ways very similar, but the ending of the second piece going to a much lower set of notes seemed very different" (Comparing A2 to A6)
- "[The first] was high-pitched, [the second] was low-pitched." (Comparing C0 and C3)
- "I felt like they had a similar pattern with how the jumps between notes." (Comparing A3 to C1)
- "Similar moving patterns. but first one started low." (Comparing A3 to C6)

While the overall clustering results varied significantly, many of the smaller clustering decisions made by the participants agreed with our system's representation.

5.3 Knowledgeable Experts Clustering

Using three faculty members from the Brigham Young University School of Music offers a way for us to assess the quality of our system from a musical expert point of view. The results of their analyses follow.

5.3.1 First Expert's Clustering

The first faculty member provided the most in-depth analysis of the three and offered two clusterings for the provided examples. In his analysis of the examples he noticed several of the attributes we had specifically attempted to optimize:

- Arch shape

- Varying range of notes
- Use of a particular interval (Perfect 4th)
- Use of repeated notes
- Chromaticism

Other identified attributes which we did not specifically target were also identified:

- Gamelan-like (A traditional Indonesian music genre)
- Use of higher notes
- Use of mordents

The first clustering he produced is based on attributes surrounding the highest note:

- Highest note repeated
- Highest note not repeated and in first three measures
- Highest note repeated in last four measures

The second clustering is based on attributes of the lowest note in a melody:

- Lowest note is repeated
- Lowest note not repeated and is first note
- Lowest note not repeated and is last note
- Lowest note not repeated and is second note
- Lowest note not repeated and is in first half of melody

The actual clusters are shown in Table 5.7.

Example Number	Ground Truth	Cluster A	Cluster B
A0	1	2	3
A2	1	3	4
A6	1	1	3
A7	1	3	5
A9	1	2	3
B3	2	2	2
B4	2	3	5
B6	2	3	5
B9	2	3	5
C0	3	1	3
C2	3	1	1
C7	3	1	1

Table 5.7: First Faculty Clustering

5.3.2 Second Expert's Clustering

The second faculty member created three clusters for his clustering. He based these clusters on the following attributes:

- Cluster 1: General arch shape, and mainly tonal in G Major/E Minor
- Cluster 2: General arch shape, examples begin sounding atonal but settle into tonal sound by their conclusion
- Cluster 3: General arch shape, highly chromatic and do not sound tonal

The actual clusters are in Table 5.8 and show that only one example differs from our system's representation. And, the "misclassified" example is classified in the next most similar group.

Example Number	Ground Truth	Clustering
A0	1	1
A2	1	1
A6	1	1
A7	1	1
A9	1	1
B3	2	2
B4	2	2
B6	2	2
B9	2	1
C0	3	3
C2	3	3
C7	3	3

Table 5.8: Second Faculty Clustering

5.3.3 Third Expert's Clustering

The third faculty member produced his clustering based on the number of accidentals used as well as which particular accidentals are used. The different groups are differentiated as follows:

- Use of accidentals $A\sharp$ and $F\sharp$
- Use of accidentals $C\sharp$ and $F\sharp$
- Use of two other accidentals
- Use of four accidentals
- use of chromatic scale

These results are listed in Table 5.9. Most notably the 5th group directly matches the C group of outputs. The other four groups correlate with the A and B examples, with the 2's and 4's contained in the A and B groups respectively.

Example Number	Ground Truth	Clustering
A0	1	1
A2	1	2
A6	1	1
A7	1	3
A9	1	2
B3	2	4
B4	2	4
B6	2	1
B9	2	3
C0	3	5
C2	3	5
C7	3	5

Table 5.9: Third Faculty Clustering

5.3.4 Faculty Results Clustering Metrics

Using the four clustering metrics we defined above, the computed metrics for each faculty member's clustering are shown in Table 5.10. Faculty member 2 has the best scores for all four metrics due to having only one example misclassified. Faculty member 3 and 1B have similar scores while 1A has the lowest overall.

	Faculty 1A	Faculty 1B	Faculty 2	Faculty 3
RI	0.66	0.74	0.87	0.75
F-Measure	0.43	0.53	0.80	0.43
NMI	0.42	0.56	0.80	0.60
Purity	0.67	0.83	0.92	0.83

Table 5.10: Faculty Results Metrics

Chapter 6

Conclusions

Producing stylistically identifiable music with GAs is hard, but our approach produces a variety of identifiable styles. The three different validations we produced show how computers and people of different abilities identified these different styles.

Our first validation method, agglomerative feature clustering, shows that our system generated three different styles based on specific musical features. The high quality of the clustering demonstrates that our multi-objective fitness function converges to the set of parameters we chose for the three different music styles. The maximum distance metric shows the greatest success in differentiating the clusters by their maximally dissimilar features.

Our second method of evaluation, using a random sampling of people and their responses, offers insight into how people experience music. There were many comparisons that people made which correlated well with the attributes we developed for the generation of music. Many of the comparisons that people made, however, were also significantly different from our system representation. This can be partly explained by a lack of repetition in our sampling. Each comparison was made only once and thus the generated distance matrix entries exhibit high variance. The responses were also telling in how normal people experience music.

Respondents specifically pointed out many of our targeted features in their responses. $KeyPrevalence$ was noticed, as many mentioned melodies having a major/minor sound versus a chromatic one. $Linearity$ was mentioned several times in relation to larger jumps, smooth lines, and a melody described as more *jumpy* than another. $SelfSimilarity$ is

hard to specifically identify in the responses. While many talked about pieces that have similar patterns of notes it is hard to qualify exactly what they described. $TopArcMelody$ was mentioned a few times about melodies going up and then down. $IntervalClass_0$ was noticed as well as a number of respondents mentioned the presence of repeated notes, but none mentioned a lack of repeated notes. $PitchRange$ seemed to be indirectly mentioned as respondents would notice that a piece was generally higher or lower than another or that a piece would end and or begin with lower or higher notes than another. Every attribute targeted in our multi-objective fitness function was mentioned in some form or another by the respondents. This evidence shows that the selected attributes are noticeable to human listeners and valid targets for music production.

There were also several attributes that respondents identified as differentiating which we did not consider in our initial implementation. The general octave in which notes occur was mentioned several times. A piece with more notes in a higher octave was considered rather different from a similar piece occurring in a lower octave. Similarly, two pieces that are somewhat close in style were considered very different if one ends on lower or higher notes than the other.

Emotionally descriptive words were also commonly used in individual responses to how they rated the pieces' similarity. "Distressed", "upbeat", and "happy" were all given as responses.

While the resulting metrics from the random sample of people were mixed in terms of clustering evaluation, they do show that many of the decisions made by individual respondents were correct. One of the larger challenges to this methodology was the single coverage of the distance matrix; had we been able to have more than one person reproduce each distance comparison, the results might have resulted in a more accurate clustering. The responses from the participants are also valuable in seeing how people perceive music differently. We found many musical attributes which we did not consider but which the participants considered significant. Thus we need to consider the set of features we analyzed,

and the set of features participants identified as a way of expanding our music composition approach.

The third method of validation offered by several School of Music faculty showed many results similar to the random sampling but also better clustering metric results. The first faculty member's clusterings did not score as high on the clustering metrics as the other faculty members', but the first faculty member's results showed several musical attributes which we had not considered. Their results and the results of the second faculty member mark a number of the same features the system targeted when producing the music. This gives evidence that the selected features are identifiable.

Overall, the three methods of validation show that the three settings of the selected features do produce three identifiable styles. These styles are identifiable to both a computer-based metric as well as by human perception. The human results also bring to light a number of features we did not consider as well as confirm that several of the selected features are perceived as important differentiating attributes.

Chapter 7

Future Work

Offering a better and more flexible platform for computer music production is the ultimate goal of this research. In this thesis we outlined our approach to overcoming the problems exhibited by other GA implementations of music production. There are a number of areas that we successfully improved upon while other challenges remain. These challenges will require broadening the scope and versatility of the music as well as improvements to the underlying GA implementation and approach.

7.1 Musical Improvements

While any length of output is possible with our system, the system still lacks the ability to produce music which would engage an audience through a significant composition. Techniques such as development of a musical idea and utilization of contrasting ideas is not possible with our current system. Producing the 30 melodies for our study took less than five minutes. Pushing our system to produce a melody a minute in length often took at least 10 minutes to produce one melody with a suitable fitness. These minute long melodies also exhibit no better than random sequences of notes that lack direction or purposeful motion. While they do score high enough on the targeted features, the diminished musical quality highlights a challenge with this approach that neglects the implementation of length normalized scores.

A major element of music that was omitted from this research is rhythm. Every melody produced by the system had the same mono-rhythmic element and each feature

extractor ignored the location in the beat structure where notes fell. As rhythm can turn an otherwise commonplace sequence of notes into music, it should be the next step of this work. Time signature, note placement in a measure and beat all impact the importance and perception of a note. Each of these issues regarding rhythm offer many ways to expand this approach to computer music composition.

Another challenge of producing music is the addition of harmonization and polyphony. Most musical styles utilize more than one note at a time and often a number of musical lines that all contribute to creating a piece of music. This also adds a great deal of complexity to any analysis as the interactions of chords, melodies and rhythms offer a multitude of elements to analyze.

Another hurdle in the perception of music is how the music is conveyed to the listener. Often some of the most musical aspects of a composition come from the performer. A live performance or recording can make a large difference in the perception of a composition. Our system does not generate many of the other important musical aspects to a performance: dynamics, accents, articulations, or tempo variances. The melodies generated have an unvarying tempo, dynamic setting and file format (MIDI). While many of these musical aspects are available in MIDI format, even with these elements a MIDI playback is often less than convincing in its musical expression.

7.2 Genetic Algorithm Improvements

Each of the elements above that outline musical improvements need to be driven by improvements in the underlying GA implementation. Feature extractors that analyze rhythm and their interaction with the notes are needed. Addition of these will improve the range of music that can be produced. As polyphony is also a desirable feature, a method of analyzing polyphonic music is also needed.

It also seems necessary to implement some elements of Genetic Programming solutions into our system. The existing system could easily be used to generate a seed melody to be

used to generate a longer musical selection through genetic programming. Examples exist of successful approaches to this [12].

7.2.1 Self-Driven Computer Music Production

During the course of this work, we determined the parameters, weights and targets for the feature extractors. This removes much of the creative responsibility from the system. An important next step for this work is to add an element of autonomy with a system that determines its own style of composition. Developing a meta-level fitness function or meta-fitness function would accomplish this.

Built on top of the existing system, a meta-fitness function would drive which features are targeted and what the target score would be. This would require careful thought to determine the (meta)fitness of a fitness function that would drive the music composition over time. The goal would not be that it necessarily converges to some sort of *best* music but that the process of music production over time would score highly. We suppose that implementing this improvement would be a large step forward in creating a truly autonomous and creative music composition system.

References

[1] Torsten Anders and Eduardo R. Miranda. A computational model that generalises Schoenberg's guidelines for favourable chord progressions. In *Proceedings of the Sound and Music Computing Conference*, pages 48–52, 2009.

[2] John Biles. Genjam: A genetic algorithm for generating jazz solos. In *Proceedings of the International Computer Music Conference*, pages 131–131, 1994.

[3] John Biles. Autonomous GenJam : Eliminating the fitness bottleneck by eliminating fitness. `http://www.ist.rit.edu/~jab/GECCO01/`, 2001.

[4] John Biles, Peter Anderson, and Laura Loggi. Neural network fitness functions for a musical IGA. In *Proceedings of the International ICSC Symposium on Intelligent Industrial Automation and Soft Computing*, pages B39–B44. ICSC Academic Press, 1996.

[5] Anthony R. Burton and Tanya Vladimirova. Generation of musical sequences with genetic techniques. *Computer Music Journal*, 23(4):pp. 59–73, 1999.

[6] David Cope. *Virtual Music: Computer Synthesis of Musical Style*. The MIT Press, 2004.

[7] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 1st edition, June 2001.

[8] Alberto Fernández and Sergio Gómez. Solving non-uniqueness in agglomerative hierarchical clustering using multidendrograms. *Journal of Classification*, 25(1):43–65, 2008.

[9] Alan Freitas and Frederico Guimarães. Melody harmonization in evolutionary music using multiobjective genetic algorithms. In *Proceedings of the Sound and Music Computing Conference*, 2011.

[10] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1st edition, January 1989.

[11] John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, Michigan, 1975.

[12] Bradley E. Johanson and Riccardo Poli. GP-music: An interactive genetic programming system for music generation with automated fitness raters. Technical Report CSRP-98-13, University of Birmingham, School of Computer Science, May 1998.

[13] Ryan McIntyre. Bach in a box: the evolution of four part baroque harmony using the genetic algorithm. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 852–857 vol.2, Jun 1994.

[14] Cory McKay. *Automatic Genre Classification of MIDI Recordings.* PhD thesis, McGill University, 2004.

[15] Cory Mckay and Ichiro Fujinaga. jSymbolic: A feature extractor for MIDI files. In *Proceedings of the International Computer Music Conference*, pages 302–305, 2006.

[16] George Papadopoulos and Geraint Wiggins. A genetic algorithm for the generation of jazz melodies. In *Proceedings of the 8th Finnish Conference on Artificial Intelligence*, volume 98, pages 7–9, 1998.

[17] Somnuk Phon-amnuaisuk, Andrew Tuson, and Geraint Wiggins. Evolving musical harmonisation. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 229–234, 1999.

[18] Tsubasa Tanaka, Takuya Nishimoto, Nobutaka Ono, and Shigeki Sagayama. Automatic music composition based on counterpoint and imitation using stochastic models. In *Proceedings of the Sound and Music Computing Conference*, 2010.

[19] Chi Lap Yip and Ben Kao. A study of musical features for melody databases. In Trevor J. M. Bench-Capon, Giovanni Soda, and A. Min Tjoa, editors, *DEXA*, volume 1677 of *Lecture Notes in Computer Science*, pages 724–733. Springer, 1999.

Appendix A

Generated Melodies

This appendix gives the complete collection of generated melodies as outlined in Section 4.1. The target attributes for each of the three clusters are found in Table 4.1.

A.1 Cluster A Melodies

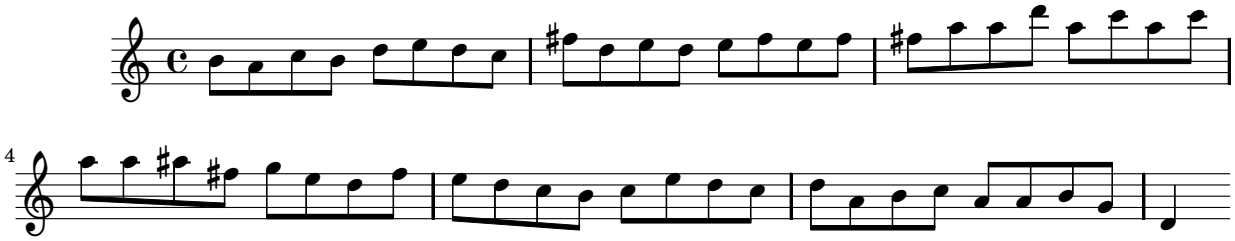


Figure A.1: Melody A0

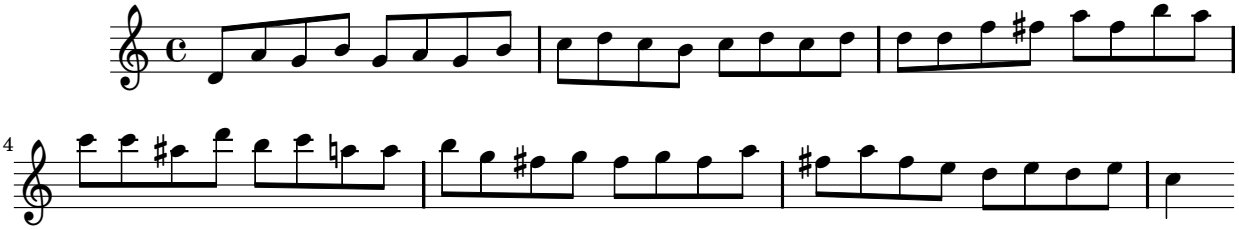


Figure A.2: Melody A1

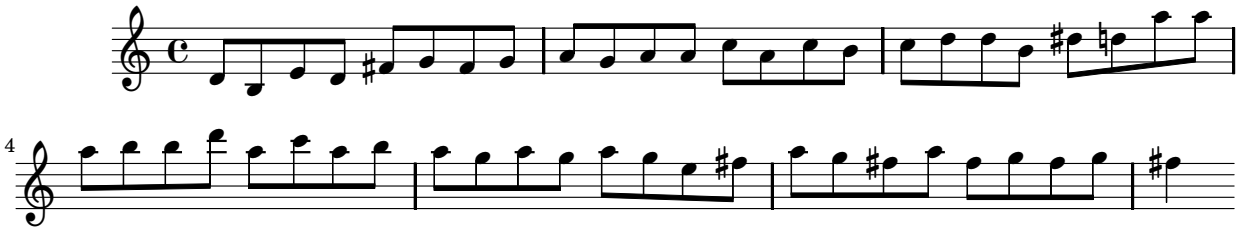


Figure A.3: Melody A2

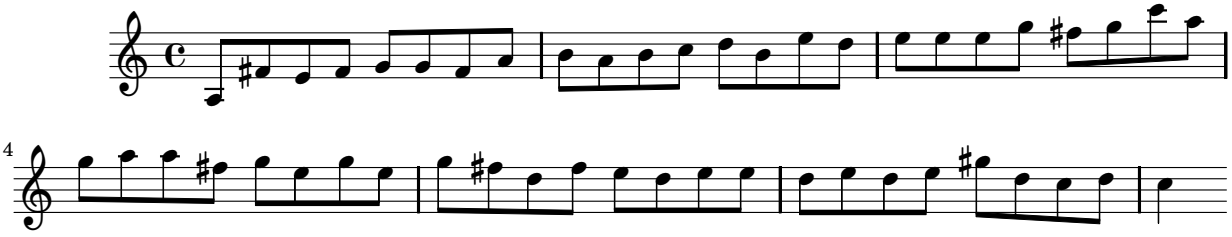


Figure A.4: Melody A3

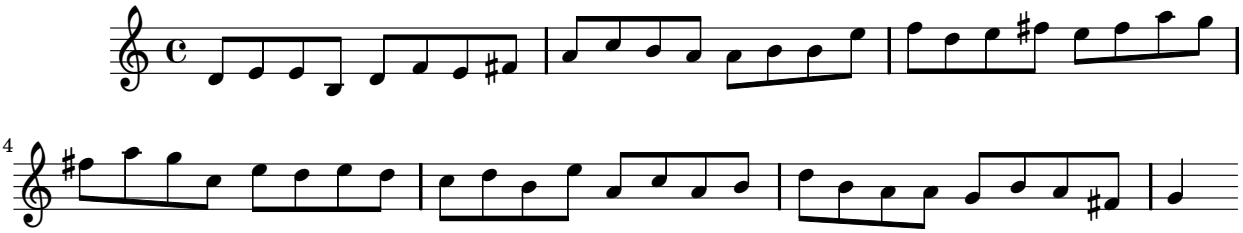


Figure A.5: Melody A4

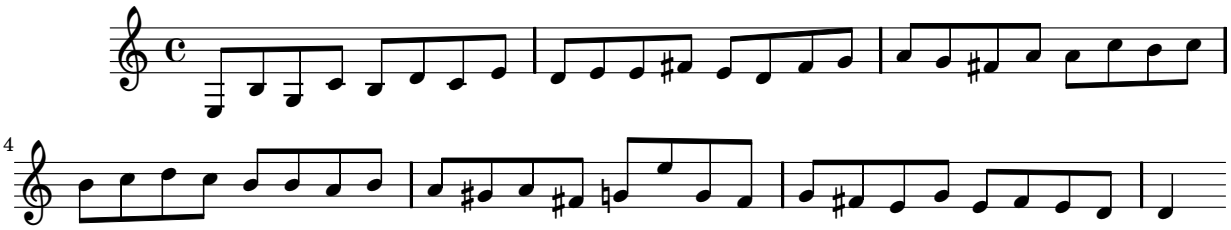


Figure A.6: Melody A5

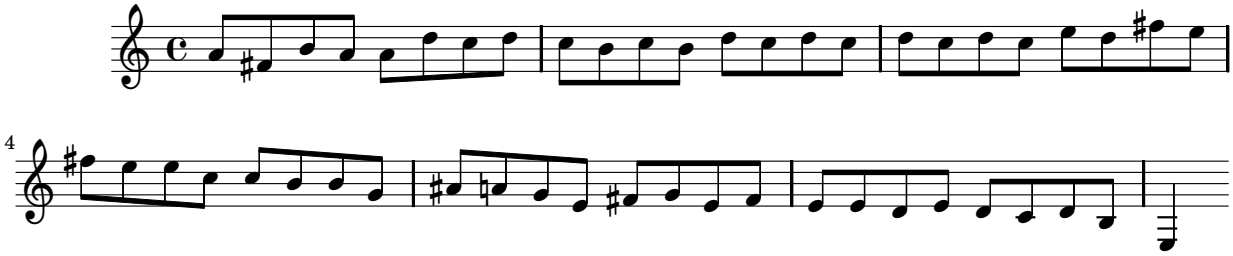


Figure A.7: Melody A6

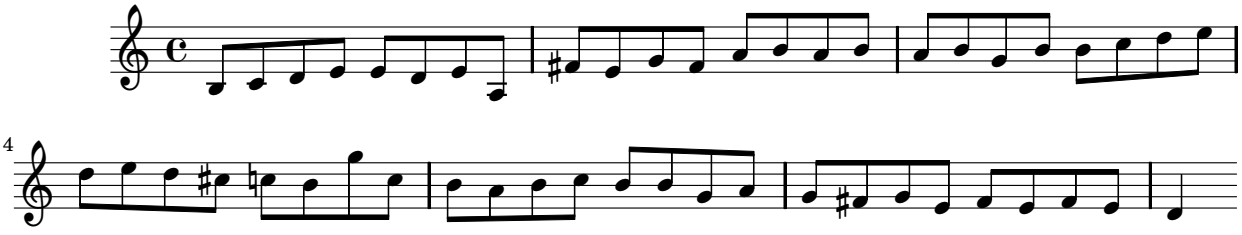


Figure A.8: Melody A7

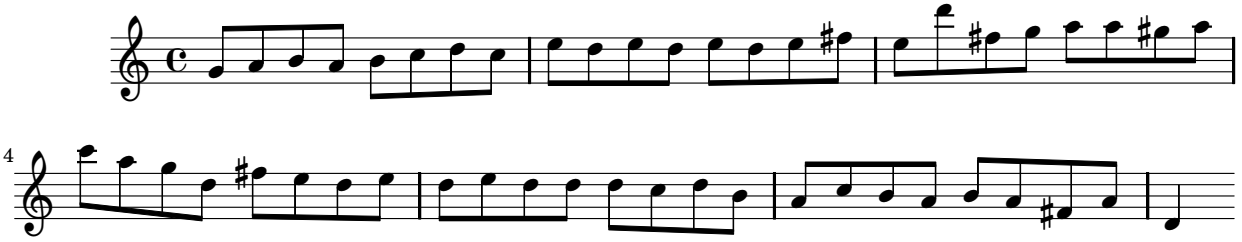


Figure A.9: Melody A8

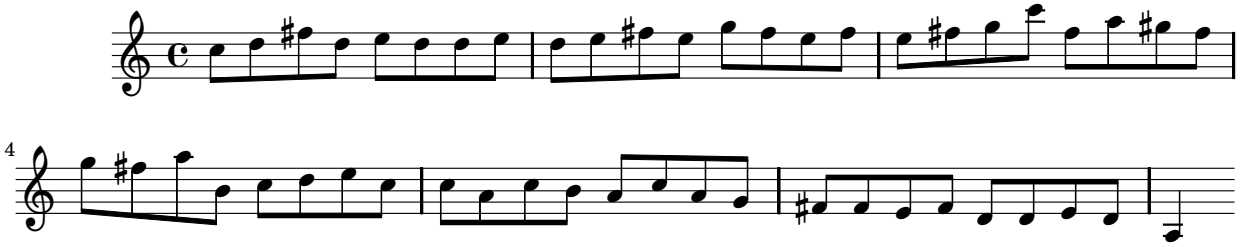


Figure A.10: Melody A9

A.2 Cluster B Melodies

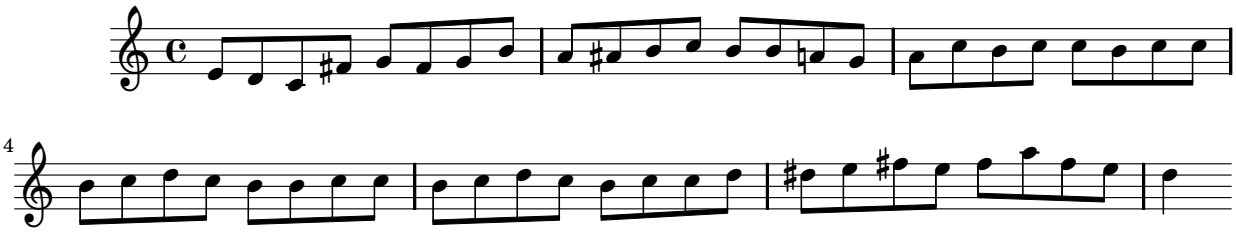


Figure A.11: Melody B0

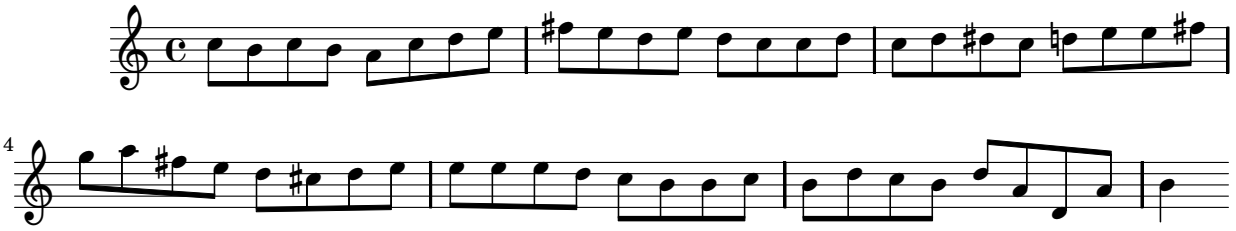


Figure A.12: Melody B1

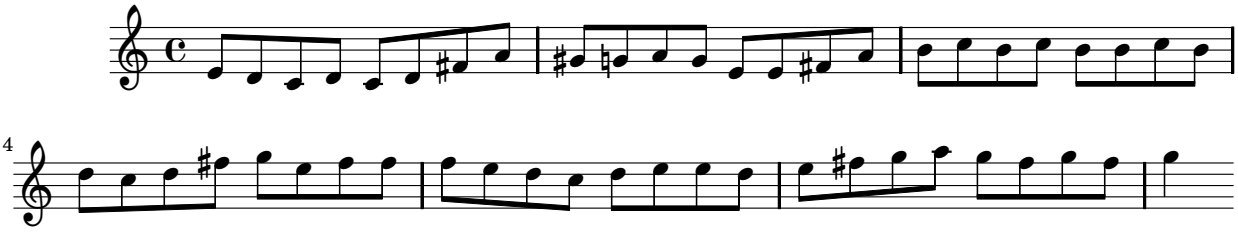


Figure A.13: Melody B2

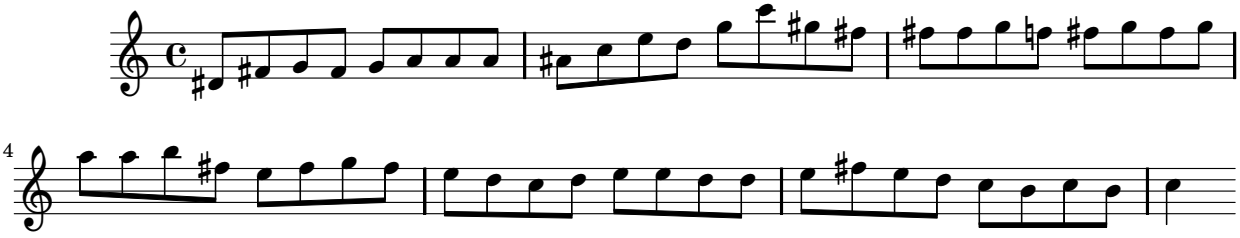


Figure A.14: Melody B3

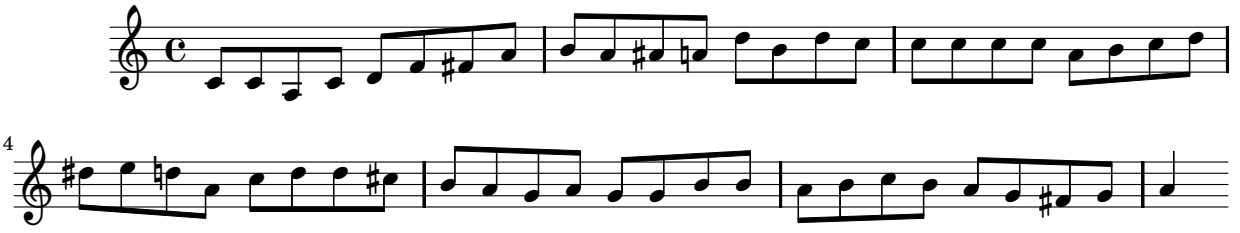


Figure A.15: Melody B4

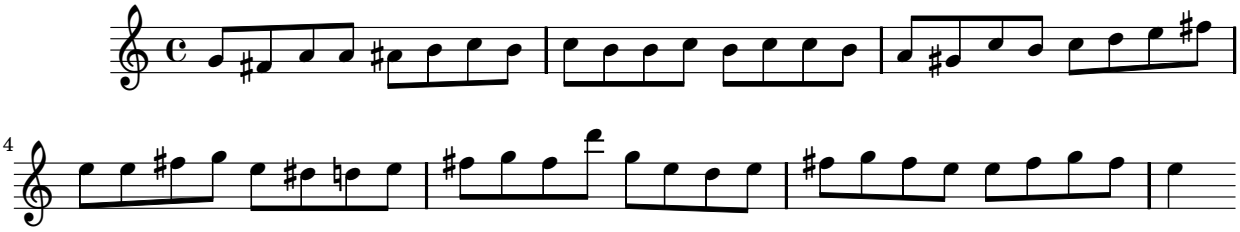


Figure A.16: Melody B5

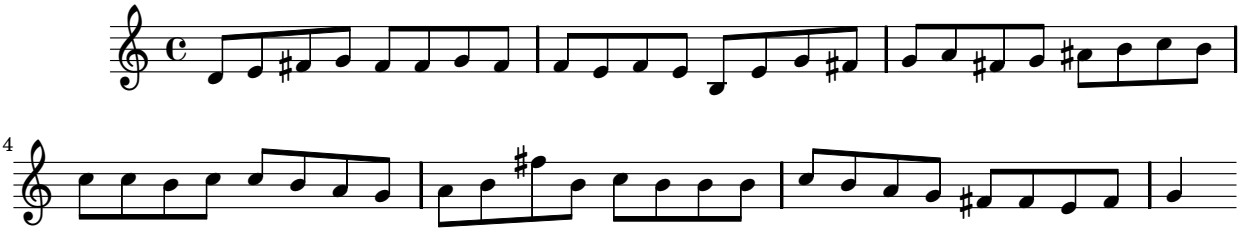


Figure A.17: Melody B6

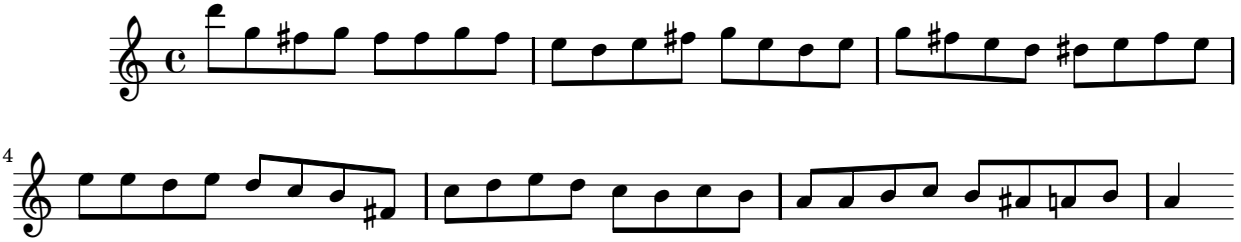


Figure A.18: Melody B7

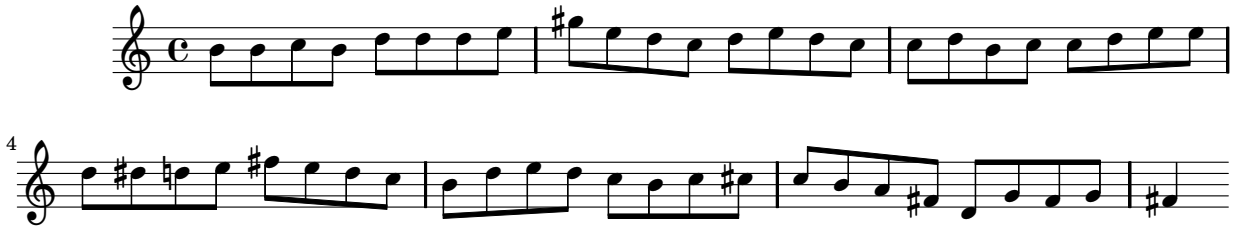


Figure A.19: Melody B8

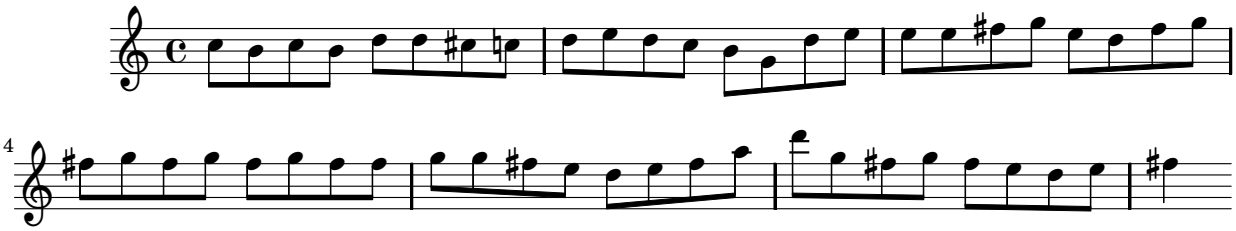


Figure A.20: Melody B9

A.3 Cluster C Melodies

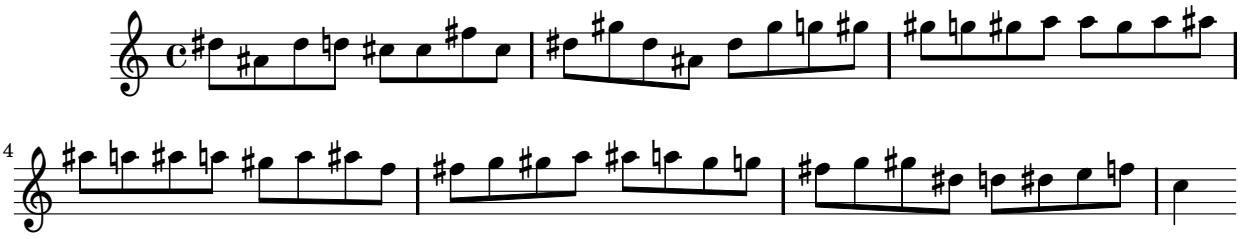


Figure A.21: Melody C0

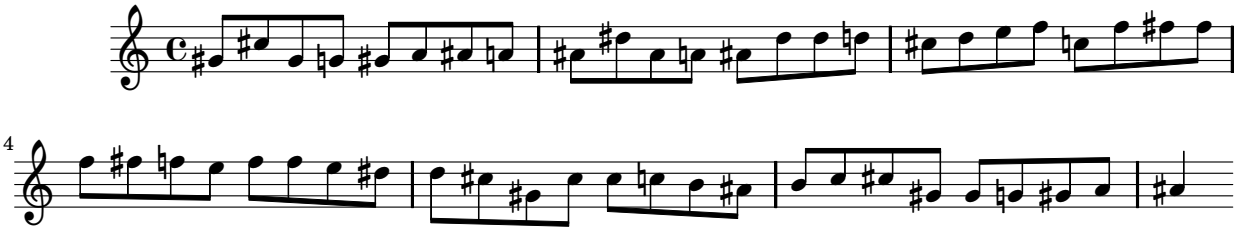


Figure A.22: Melody C1

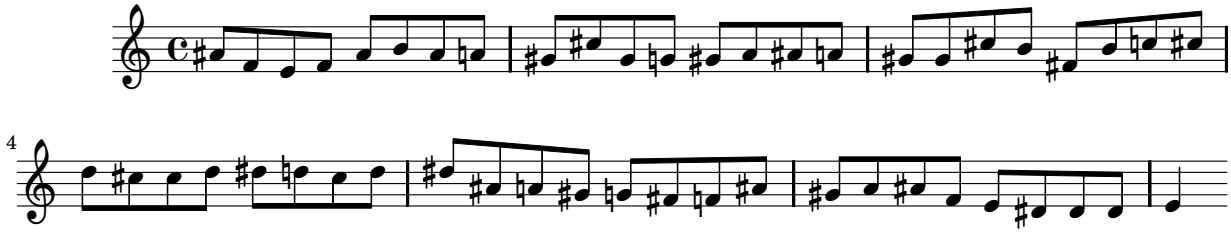


Figure A.23: Melody C2

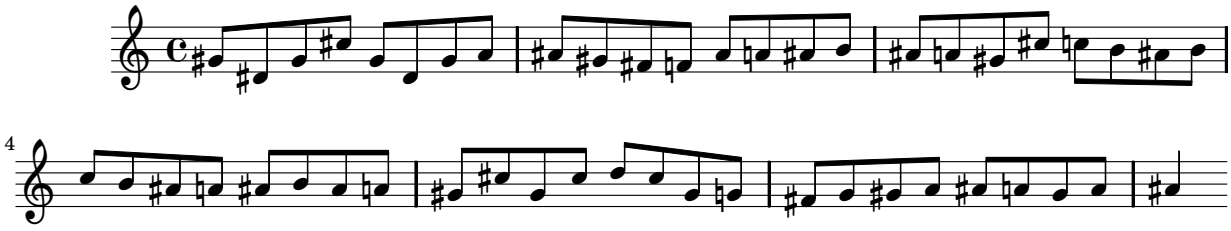


Figure A.24: Melody C3

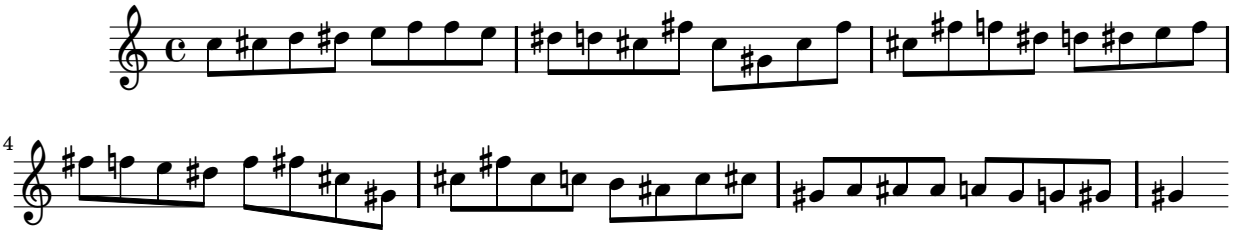


Figure A.25: Melody C4

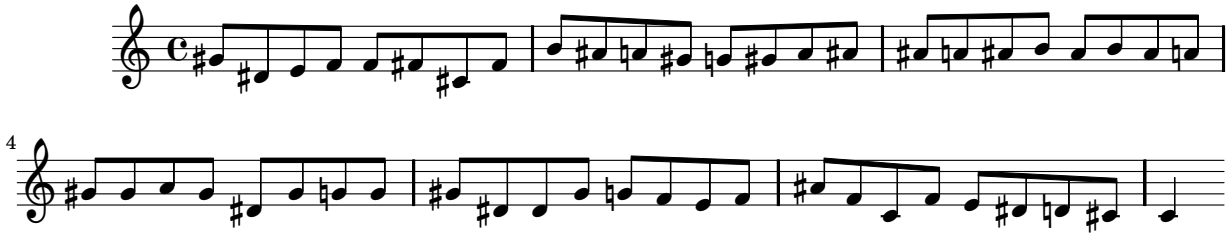


Figure A.26: Melody C5

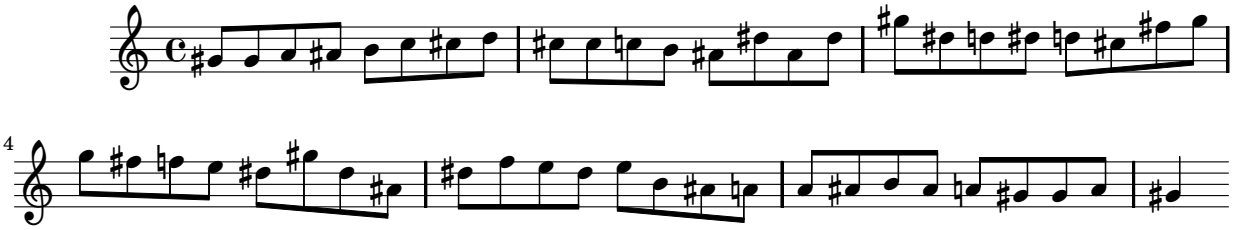


Figure A.27: Melody C6

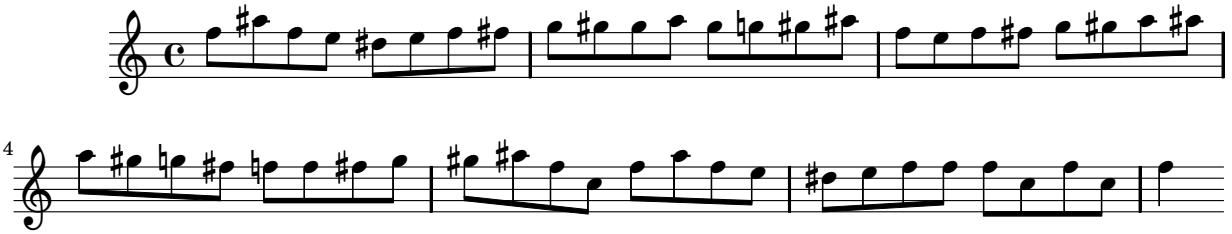


Figure A.28: Melody C7

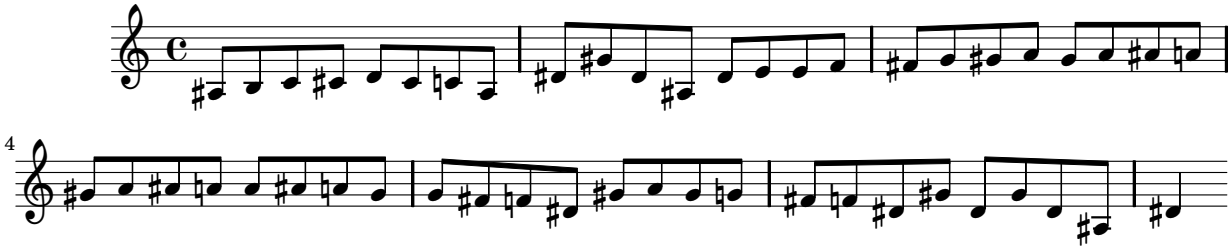


Figure A.29: Melody C8

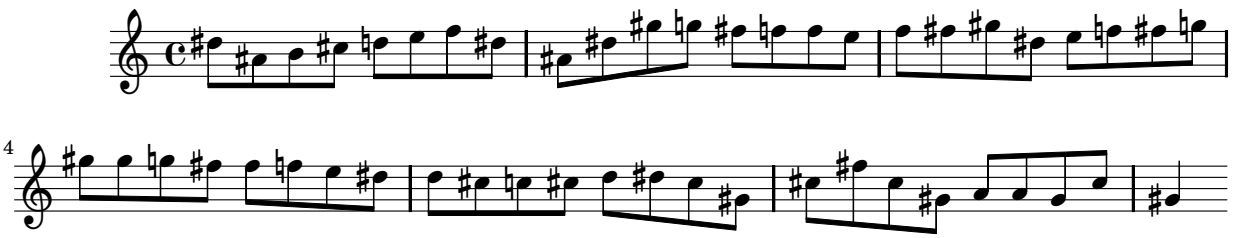
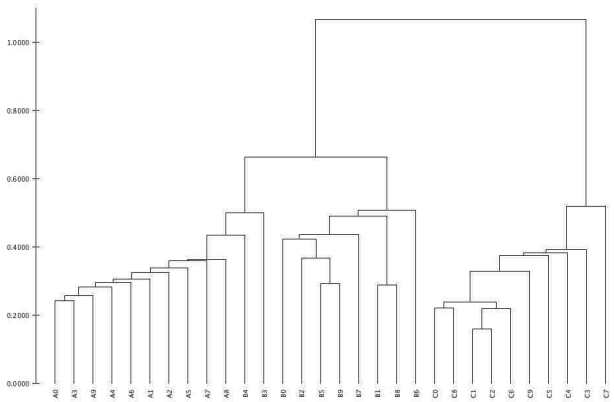


Figure A.30: Melody C9

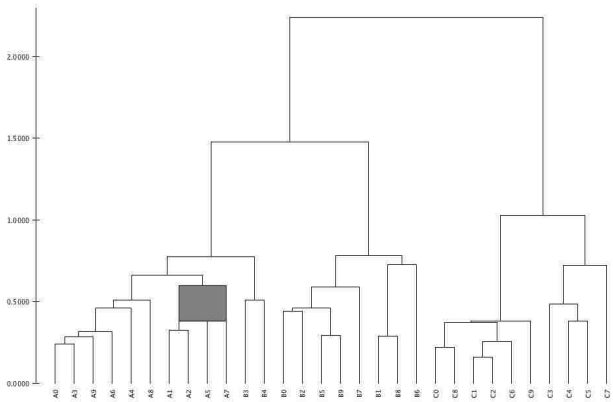
Appendix B

Dendrograms

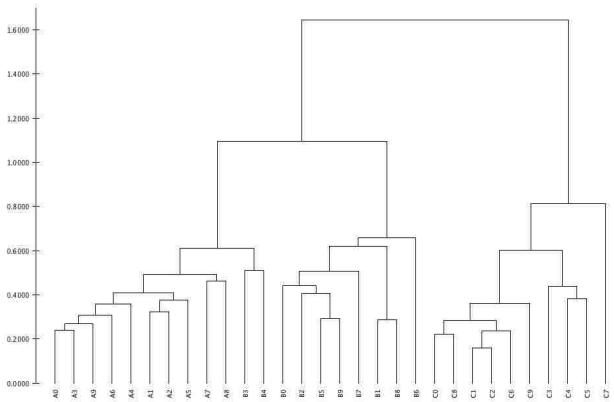
This appendix gives the resulting dendrograms from our agglomerative feature clustering validation method. Each group of figures contains the dendrograms according to a particular distance metric while each individual figure is captioned according to the clustering algorithm used to produce the dendrogram.



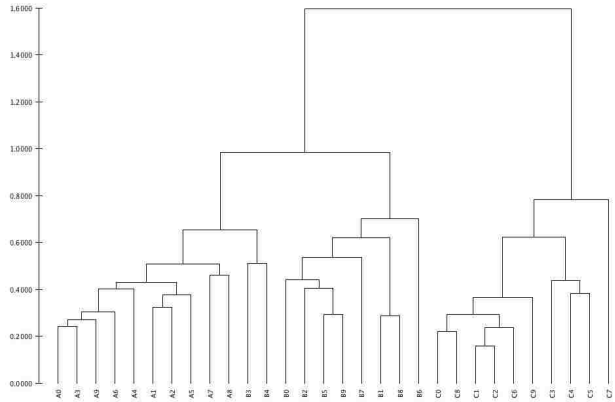
(a) Single Linkage



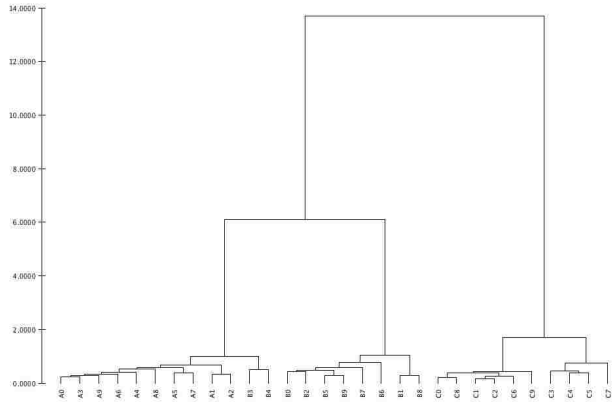
(b) Complete Linkage



(c) Unweighted Average

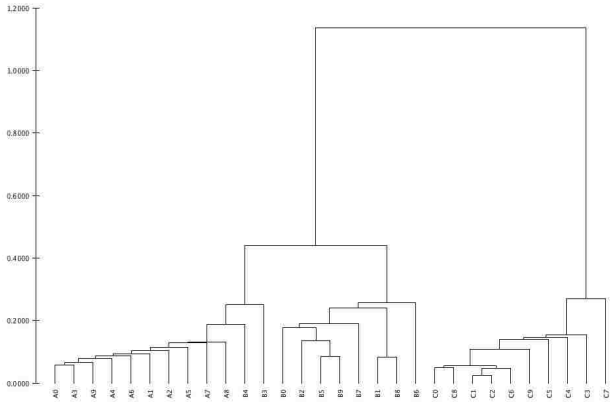


(d) Weighted Average

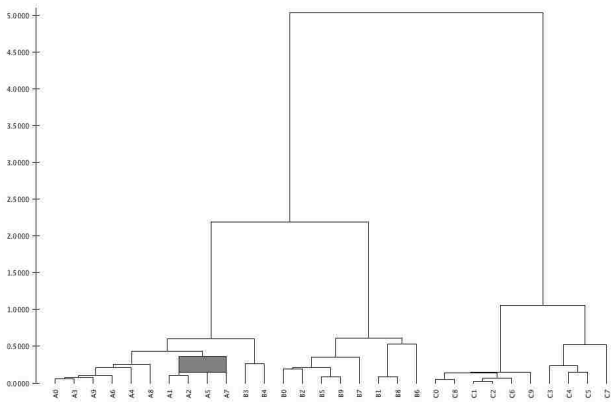


(e) Joint Between-Within

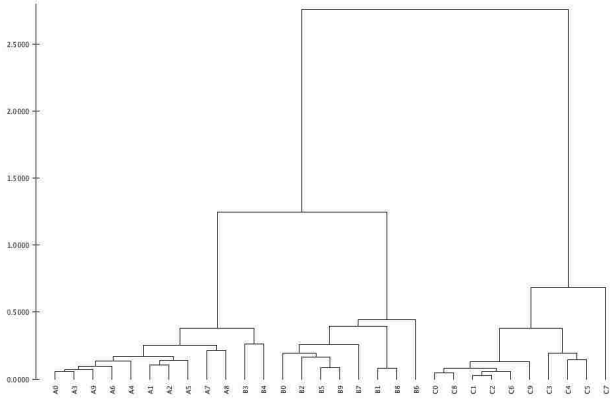
Figure B.1: Euclidean Dendrograms



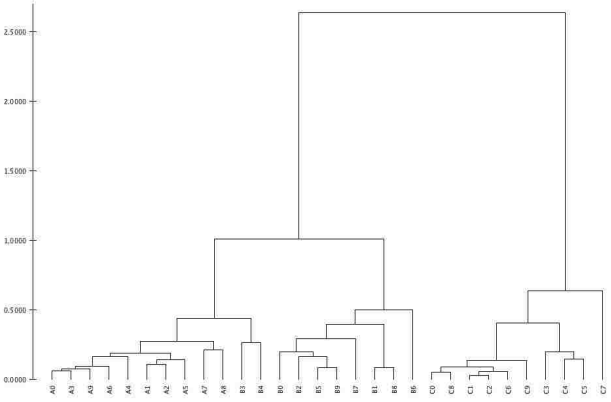
(a) Single Linkage



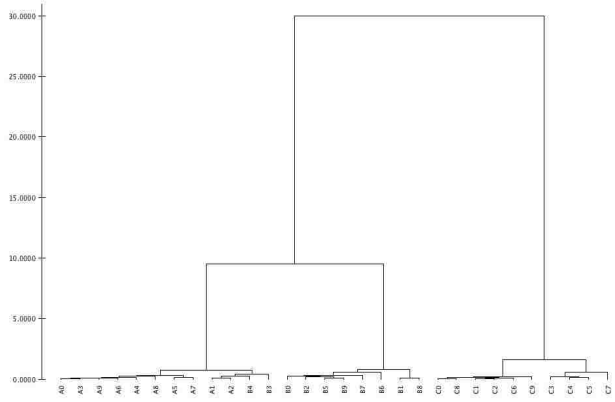
(b) Complete Linkage



(c) Unweighted Average

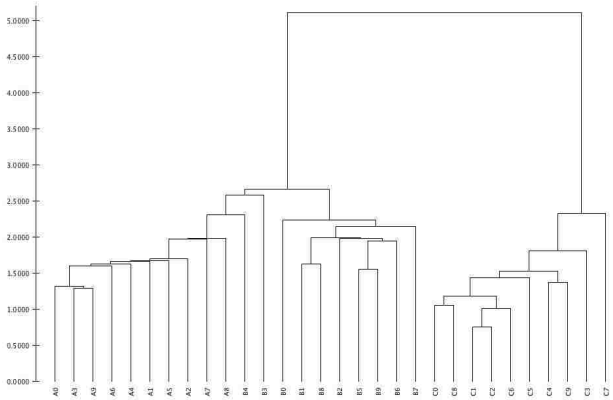


(d) Weighted Average

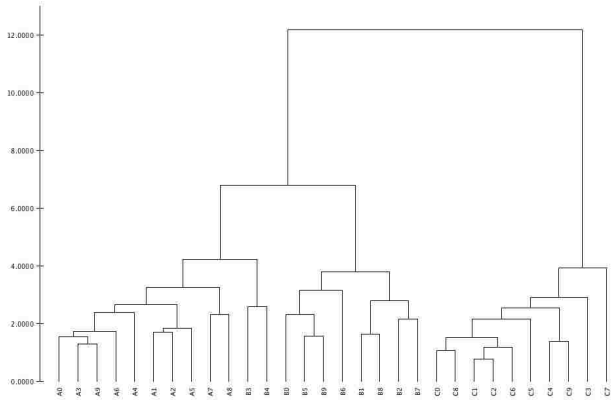


(e) Joint Between-Within

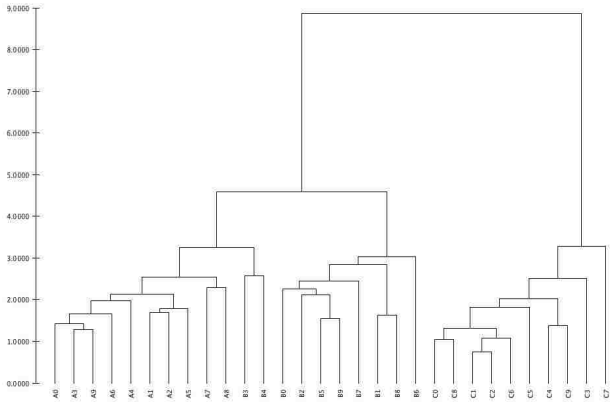
Figure B.2: Euclidean Squared Dendrograms



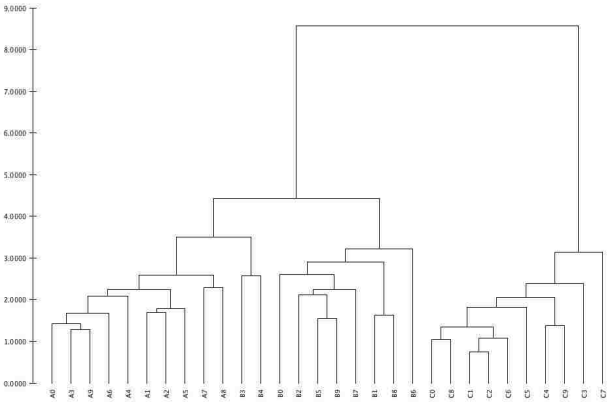
(a) Single Linkage



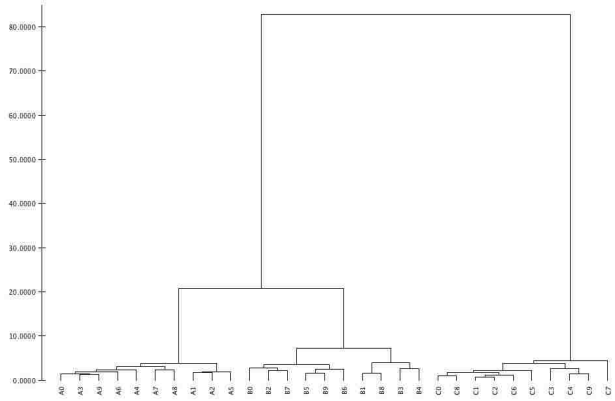
(b) Complete Linkage



(c) Unweighted Average

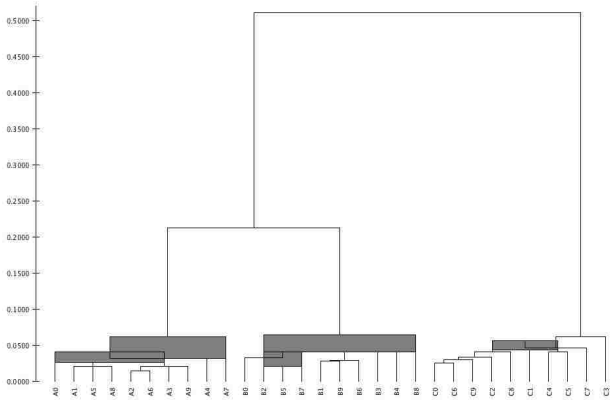


(d) Weighted Average

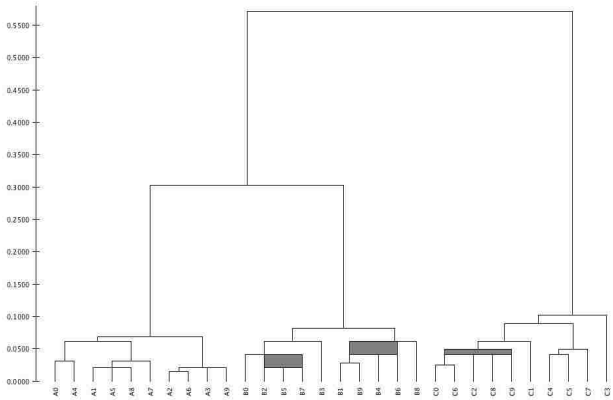


(e) Joint Between-Within

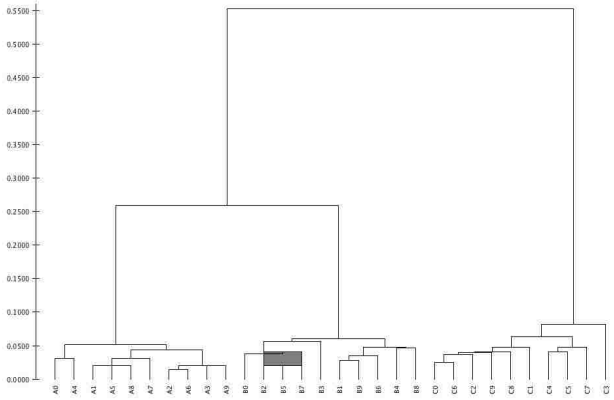
Figure B.3: Manhattan Dendrograms



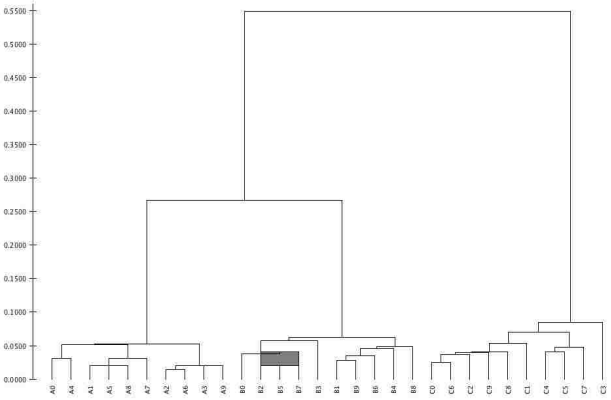
(a) Single Linkage



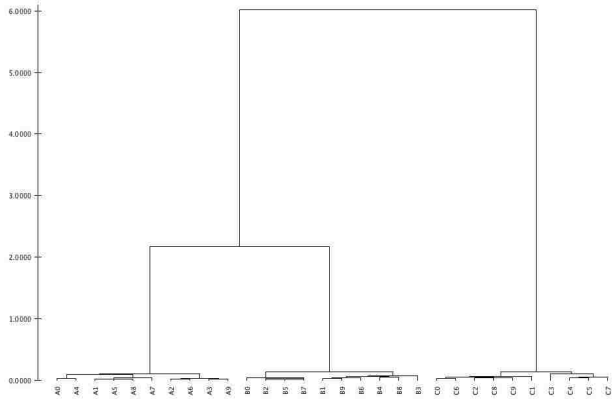
(b) Complete Linkage



(c) Unweighted Average

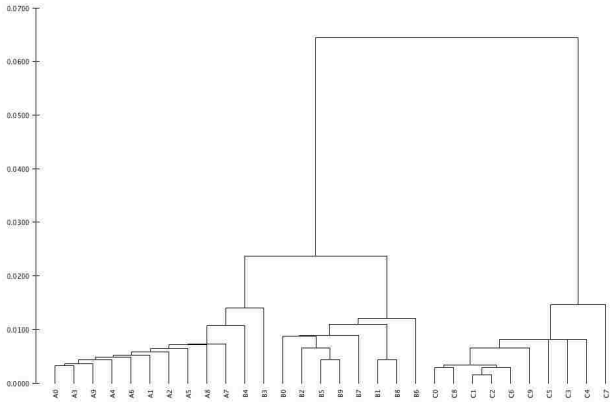


(d) Weighted Average

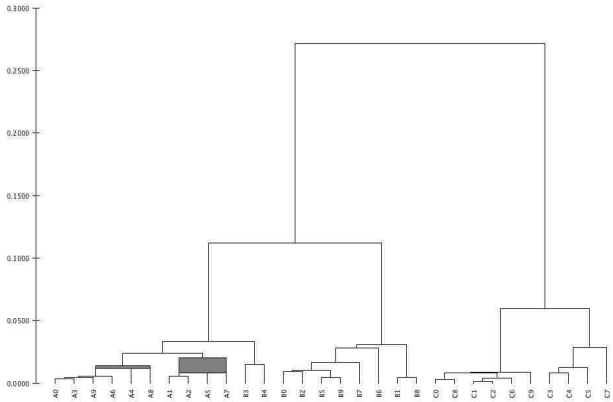


(e) Joint Between-Within

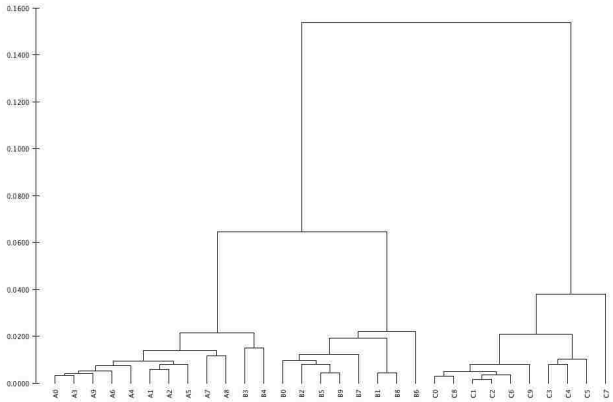
Figure B.4: Maximum Dendrograms



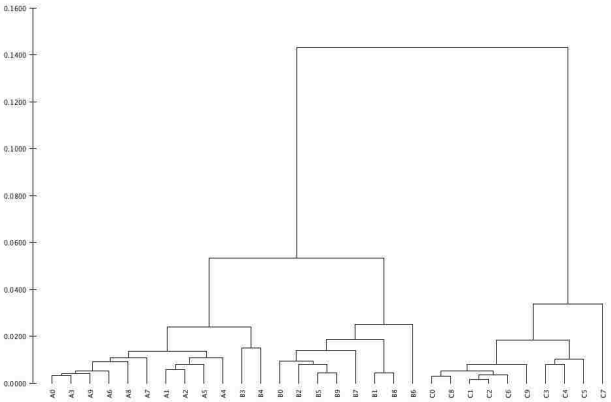
(a) Single Linkage



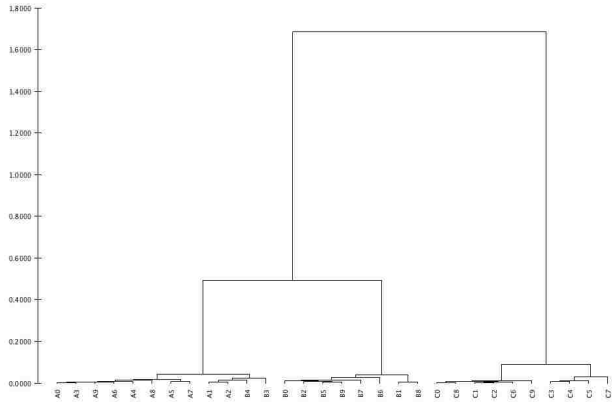
(b) Complete Linkage



(c) Unweighted Average

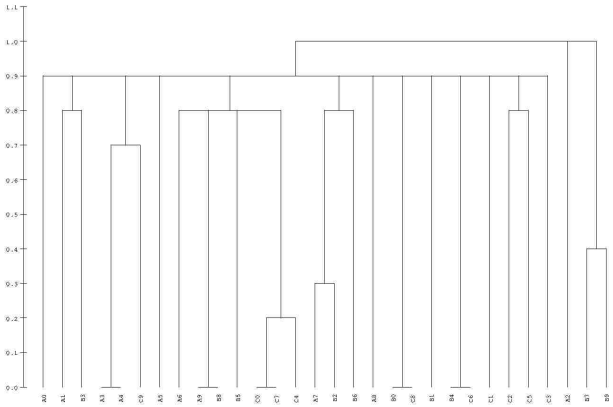


(d) Weighted Average

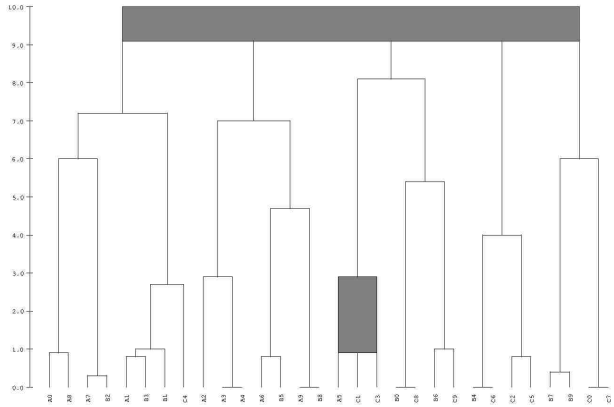


(e) Joint Between-Within

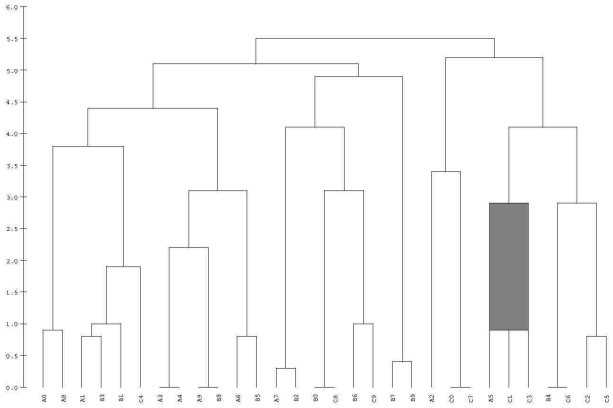
Figure B.5: Cosine Similarity Dendrograms



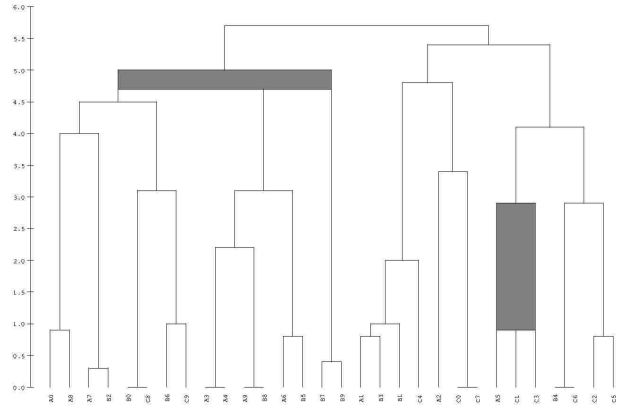
(a) Single Linkage



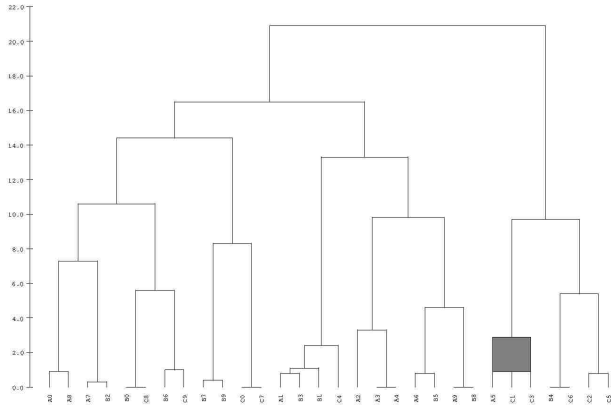
(b) Complete Linkage



(c) Unweighted Average



(d) Weighted Average



(e) Joint Between-Within

Figure B.6: Human Distance Metric Dendrograms