Algorithmically Flexible Style Composition Through Multi-Objective Fitness Functions

Skyler Murray and Dan Ventura
Department of Computer Science
Brigham Young University
skyler.murray@byu.edu, ventura@cs.byu.edu

Abstract

Creating a musical fitness function is largely subjective and can be critically affected by the designer's biases. Previous attempts to create such functions for use in genetic algorithms lack scope or are prejudiced to a certain genre of music. They also are limited to producing music strictly in the style determined by the programmer. We show in this paper that *musical feature extractors*, which avoid the challenges of qualitative judgment, enable creation of a multi-objective function for direct music production. The main result is that the multi-objective fitness function enables creation of music with varying identifiable styles. To demonstrate this, we use three different multi-objective fitness functions to create three distinct sets of musical melodies. We then evaluate the distinctness of these sets using three different approaches: a set of traditional computational clustering metrics; a survey of non-musicians; and analysis by three trained musicians.

Introduction

Computational music composition is a challenging area of computational creativity that has spawned a variety of approaches to the problem of automatic generation of music. However, music theory comprehends a complexity that is in some ways difficult to formalize, making 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 (Cope 2004). Other examples include Anders and Miranda (2009) demonstrating an effective method for producing chord progressions that follow established rules and Tanaka et al. (2010) encoding the rigorous rules of two-part counterpoint into stochastic models that produce convincing results.

While many successful approaches exist, Genetic Algorithms (GA) offer perhaps the greatest flexibility for producing varied musical outputs due to the abstract nature of the fitness function and generality of the genomic representation. Freitas and Guimaraes (Freitas and Guimarães 2011) 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 harmonizations that utilized one of two different styles—simplicity or dissonance.

Genetic algorithms can be successfully applied to the musical domain, but implementing an effective fitness function remains a challenge (Burton and Vladimirova 1999) because quantifying how *good* a piece of music is remains largely subjective—a major hurdle for the domain. Currently the two most common approaches are human-in-the-loop and algorithmic.

Interactive Genetic Algorithms (IGA) incorporate human input as the fitness function but suffer from throughput issues—the fitness bottleneck (Biles 1994). 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 (1994) on an evolutionary composition system—GenJam—produces improvisational jazz lines by employing interaction with a human rater, and he agrees that this approach leads to low throughput. He attempts to implement a neural network (Biles, Anderson, and Loggi 1996) to overcome this challenge but is unable to produce the same quality of results achieved by the human rater. Biles' later efforts to move away from IGA(Biles 2001) suggest that another approach is desirable.

An automated fitness function increases throughput but has traditionally limited evaluation quality due to the narrow scope of most implemented fitness functions. Whether the fitness function is designed to look for specific 4-part harmony rules (McIntyre 1994) 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 propose a system that addresses these challenges and allows a genetic algorithm to flexibly produce multiple unique styles by employing a set of feature extractors. Individual extractors analyze 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 uses a weighted combination of the feature extractors to produce a fitness score which the GA uses to drive the evolutionary model. This system produces distinct musical styles determined by the fitness function weights.

Related Work

Genetic Algorithms

Genetic Algorithms (GA) (Goldberg 1989; Holland 1975) 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 (Deb and Kalyanmoy 2001).

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 choses individuals who will populate a *mating 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 (Burton and Vladimirova 1999). See Algorithm 1.

The wide applicability of GAs (Deb and Kalyanmoy 2001) shows promise for applications in music generation. Phon-Amnuaisuk, Tuson and Wiggins (1999) 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* (1994). Without this knowledge it is difficult to produce meaningful music.

Freitas and Guimarães (2011) demonstrate the power of using musically meaningful mutations, implementing musical versions of crossover and mutation as well as 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.

Algorithm 1 Genetic Algorithm

```
Initialize population
while not done do
    Calculate fitness for all individuals
    Order individuals by fitness
    Create probabilistic 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
Output $m$ individuals with fitness $> T$
```

Fitness Functions

At the core of any GA is a fitness function which drives the evolutionary process 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. Convergence to a useful solution depends critically on the fitness function's representation of genomic quality, but music is an ill-defined concept that is difficult to qualify precisely.

However, when the scope of the evaluations is limited, a fitness function can effectively drive a population to converge on quality solutions. Freitas and Guimarães (2011) 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 their level of dissonance. After many generations of harmonizations, the surviving individuals exhibit traits which are scored highly by at least one of the these fitness functions, demonstrating how two different fitness functions enable the generation of two types of output.

Feature Extractors

Selection of appropriate features for musical representation facilitates many computational approaches to music processing in the broadest sense. Examples include Yip et. al (1999) showing how extracting features from music is useful in cataloging melodies; McKay (2004) creating a successful music genre classification system based on music feature extraction (with the software later published an open-source library (Mckay and Fujinaga 2006)); and Brown (2004) evaluating combinations of musical features and generative methods for producing aesthetically pleasing melodies.

Methodology

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

- Uses a set of musical feature extractors
- Drives the evolutionary process with a linear, multi-objective fitness function that employs the feature extractors
- Produces music of varying, yet identifiable styles

Genetic Algorithms

Our approach uses variations on several aspects of the traditional GA while keeping the general algorithm intact (see Algorithm 1). instead of representing individuals as a binary string, our implementation use a string of note names. Larger departures from the traditional GA implementation involve the crossover and mutation operators, for which we implement musically meaningful operators. The fitness function implementation is also a departure from common approaches to music. These differences are discussed in more detail below.

Musically Meaningful Operators

Many implementations exist in the literature for musically meaningful operators (Johanson and Poli 1998; Phon-amnuaisuk, Tuson, and Wiggins 1999; Freitas and Guimarães 2011; McIntyre 1994; Papadopoulos and Wiggins 1998). Many of these approach the problem by implementing additional operators beyond the traditional crossover and mutation. However, here we focus on implementing just the traditional crossover and mutation operators in a musically meaningful way that will enable production of stylistically varied music.

We implement a standard one-point crossover where the splitting point in two individuals is randomly chosen between two notes. We implement mutation as an alteration to a single note by probabilistically altering its pitch up or down. These changes to the standard GA operators will allow for the mutation and crossover phases of our GA to happen without significant computational overhead.

Feature Extractors

A set of feature extractors provide the inputs to the fitness function from which each individual's fitness score is calculated. 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 a very simple example, a feature might compute the percentage of notes from the musical key of G major, returning the percentage as the output.

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 we implement follow below:

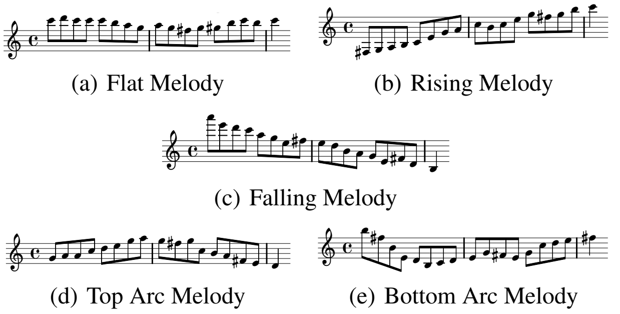
- **Self-Similarity:** Measures how often repeating interval sequences occur in I and uses this as a measure of self-similarity—if the same interval occurs often, the piece is more self-similar than if many different intervals occur less frequently.

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

  where

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

  and S is the set of all interval sequences of length 2 that appear in I, and $\mathit{count}_s(I)$ is the number of times interval sequence s occurs in I.



(a) Flat Melody

(b) Rising Melody

(c) Falling Melody

(d) Top Arc Melody

(e) Bottom Arc Melody

Figure 1: The five types of melody shapes used to parameterize the Shape_{type} feature extractor.

- **Melody Shape:** A set of five functions parameterized by a particular melody shape. The melody shape parameter can take one of five type values: Flat, Rising, $\mathit{Falling}$, TopArc, $\mathit{BottomArc}$. These shapes are illustrated in Figure 1.

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

  where $m = i_n - i_0$ is the overall slope of I, ϵ is the the mean squared error (computed using linear regression) between the shape type and I, $m_{max} = \frac{\mathit{NoteRange}}{|I|}$ and $\mathit{NoteRange} = 49$.

- **Linearity:** Measures how angular the notes in I are. 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.

  $$\mathit{Linearity}(I) = \frac{\alpha S(I)^2}{\alpha S(I)^2 + 1}$$

  Here, $S(I)$ is an approximation for the second partial derivative, similar to a Laplacian kernel:

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

- **Key Prevalence:** 12 functions for each possible key center. Measures the proportion of notes from I that represent that key.

  $$\mathit{KeyPrevalence}_j(I) = \frac{|K_j|}{|I|},$$

  where, $K_j = \{i \in I | i \in \mathit{Key}_j\}$, $1 \leq j \leq 12$, and Key_1 is C Major, Key_2 is G Major...Key_{12} is F Major.

- **Range of Pitch:** Scores how much of the full range of notes 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 the percentage of notes in the four-octave range covered by I. We use a non-linear scaling factor γ that weights the use of the first two octaves more heavily than the third and fourth.

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

- **Interval Class Prevalence:** Similar to the $KeyPrevalence()$ features. 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, $0 \leq j \leq 11$.

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

Scoring Features

Because evolutionary pressure maximizes fitness, we require a way to target any specific feature score (rather than just driving the population to 1 for any given feature). We use a wrapper function that compares a target value t to the value of the feature extractor $e(I)$ and returns a value in the range $[0, 1]$, with a higher value indicating a better match:

$$FeatureScore(t, e(I)) = \frac{-1}{(x(t) - t)^2} (e(I) - t)^2 + 1$$

where

$$x(t) = \begin{cases} 1, & \text{if } t < 0.5 \\ 0, & \text{if } t \geq 0.5 \end{cases}$$

Multi-Objective Fitness Function

The multi-objective fitness function provides a flexible framework for producing 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.

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

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 $\cup$ individual with bestfitness $> T$
return C
```

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.

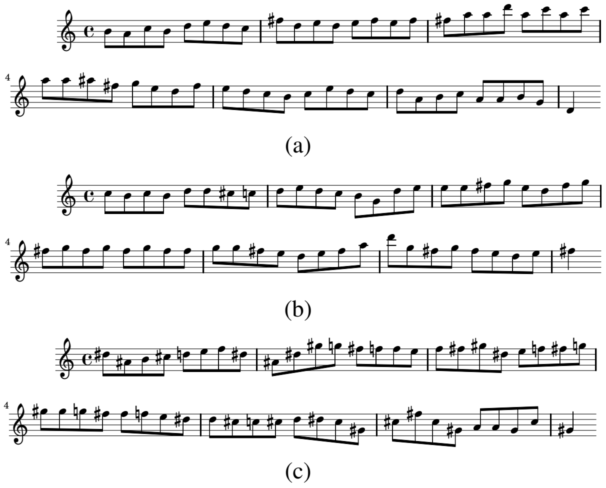
Results

We used three different multi-objective fitness functions, f_{A}, f_{B} and f_{C}, chosen so that A and C represent quite different musical styles with B representing a similar style to A (Table 1 gives the target values and weights for the feature extractors for the three styles). We then generated 10 different melodies for each of the three styles (referred to here as A0,...,A9, B0,...,B9, C0,...,C9) using Algorithm 2 and a genetic population of 40 individuals initialized with random notes in the four octave range. Figure 2 shows representative examples of each style and all 30 generated melodies may be listened to at `http://axon.cs.byu.edu/styleGeneration`.

We then evaluated these three sets of melodies for intra-group cohesion and inter-group distinction, using three different methods of evaluation: traditional agglomerative clustering, non-musician survey responses and analysis by trained musicians. Finally, for all three evaluation methods, we evaluate the quality of the resulting clustering using four common clustering metrics: purity, Rand index, F-measure and normalized mutual information.

	Style A		Style B		Style C	
	Weight	Target	Weight	Target	Weight	Target
$SelfSimilarity$	20	0.05	20	0.05	20	0.2
$Shape_{TopArc}$	20	0.8	20	0.6	20	0.8
$IntClassPrev_0$	10	0.05	10	0.05	10	0.0
$IntClassPrev_5$	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 1: Features, weights and targets for three musical styles. Style 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, and is only somewhat linear. Compare with the example score in Figure 2(a). Style B is similar to Style 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 fewer large jumps in opposite directions and more consistent use of the same interval. Compare with the example score in Figure 2(b). In contrast, Style C has a raised target for self similarity, which favors music with more frequent and common interval patterns. The changes to the $IntClassPrev_0$ and $IntClassPrev_5$ targets produce melodies with fewer repeated notes and a predominant use of the Major 4th interval. The reduction in the target score for pitch range results in a smaller range of notes, and lowering the target for $KeyPrevalence$ drives the majority of the notes to be outside of G Major. Compare with the example score in Figure 2(c).


(a)

(b)

(c)

Figure 2: Representative examples of music composed in (a) style A, (b) style B and (c) style C.

Agglomerative clustering

Agglomerative clustering requires the specification of both a distance metric and a technique for computing cluster distances. We used five common distance metrics (L_1-norm, L_2-norm, L_4-norm, L_∞-norm and *cosine similarity*) and five common clustering techniques (*single linkage*, *complete linkage*, *weighted average*, *unweighted average* and *joint between-within*) to compute 25 different agglomerative clusterings. All 25 correspond very closely to the three styles represented by the three multi-objective fitness functions—six of the clusterings were perfect (L_1-norm with joint between-within and L_∞-norm with all five cluster techniques), while the other 19 grouped only two selections (B3 and B4) incorrectly. In addition, styles A and B were consistently clustered together before style C, as expected. See Figure 3 for a representative example dendrogram.

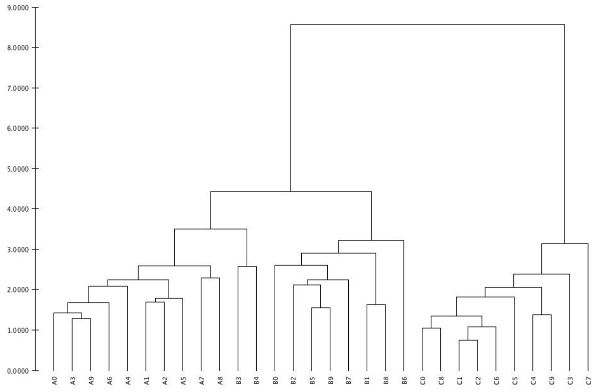


Figure 3: Agglomerative clustering of results using the L_1-norm with *weighted average* to compute inter-cluster distances. Note that at the three-cluster threshold, three distinct styles are represented that correspond very closely to the three multi-objective fitness functions use to generate the selections, with only selections B3 and B4 grouped incorrectly. Also note that at the two-cluster threshold, styles A and B are clustered together and remain distinct from C.

Non-musician survey

We asked non-musician respondents to listen to pairs of melodies and score their similarity on a scale of 0 to 10. We also asked them to describe why they scored the pairs the way they did. Using the similarity scores collected in the survey, we generated a 30×30 distance matrix for all the melodies in all three style groups. This distance matrix was then used as the distance metric to produce five additional agglomerative clusterings using the five clustering techniques mentioned above.

In this case, the clusterings were much less consistent and did not agree nearly as well with the three generative styles. There are likely several reasons for this, including the fact that respondents did not have significant musical training, the fact that people will naturally perform contradictorily on this type of task and the fact that the small number of survey participants means that each distance matrix entry was generated, on average, from a single response, likely resulting in an undesirable level of variability in the matrix elements. Still, even with all of these issues, some of the underlying structure is still discernible in these human-based clusterings (see Figure 4).

Several comments made by respondents demonstrate that in their analyses they were using features similar to those implemented in the system:

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

Of course, other comments indicate that the respondents were looking at things quite differently (and possibly sometimes reveal a lack of musical maturity):

- "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)

Musician analysis

We chose a random subset of the 30 melodies: five of stye A, $\{A0, A2, A6, A7, A9\}$; four of style B, $\{B3, B4, B6, B9\}$; and three of style C, $\{C0, C2, C7\}$; and asked three music faculty members who routinely analyze music, to cluster the pieces. Specifically, each faculty member was asked to

1. Listen to and study the scores for each selection
2. Analyze and list distinctive traits for each selection
3. Group the selections into a number of different groups based on common traits (we did not tell them how many groups)
4. List the traits that differentiate each group from the others

Musician 1 identified several of the features included in our system: arch shape, varying range of notes, use of a particular interval (Perfect 4th), use of repeated notes and chromaticism; they also identified several other features not explicitly included in the system: gamelan-like (a traditional Indonesian music genre), use of higher notes, use of mordents. Their response also offered two different clusterings. The first consists of three clusters and is based on attributes of the highest note with the clusters described as: highest note repeated, highest note not repeated and in the first three measures, highest note repeated in last four measures and is composed as follows: $\{A6, C0, C2, C7\}$, $\{A0, A9, B3\}$ and $\{A2, A7, B4, B6, B9\}$. The second is composed of five clusters and is based on attributes of the lowest note: lowest note 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, and is composed as follows: $\{C2, C7\}$, $\{B3\}$, $\{A0, A6, A9, C0\}$, $\{A2\}$ and $\{A7, B4, B6, B9\}$. These clusterings differ significantly from the "ground truth", but they also share much common structure.

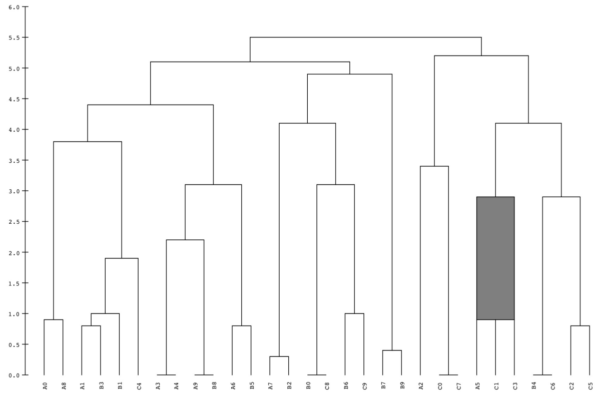


Figure 4: Agglomerative clustering of results using a distance matrix constructed from the survey responses of non-musician subjects with *unweighted average* to compute inter-cluster distances. Note that at the three-cluster threshold, there is less agreement with the styles represented by the multi-objective fitness functions used to generate the selections. Still, however, each of the three clusters is populated predominantly by a single generated style: 7/12 style A in the leftmost cluster, 5/8 style B in the center cluster and 7/10 style C in the rightmost cluster.

Musician 2 clustered the songs into three groups, describing them as follows. Group1 members have a general arch shape and are mainly tonal in G Major/E Minor; Group2 members have a general arch shape and begin atonally but settling into tonal sounds by their conclusion; Group3 members have a general arch shape, are highly chromatic and do not sound tonal. The actual clustering given by Musician 2 is $\{A0, A2, A6, A7, A9, B9\}$, $\{B3, B4, B6\}$ and $\{C0, C2, C7\}$. Note the marked similarity between this clustering and the "ground truth", with only a single song placed differently.

Musician 3 clustered based on number and placement of accidentals, giving five clusters described as: use of accidentals A♯ and F♯, use of accidentals C♯ and F♯, use of two other accidentals, use of four accidentals, use of chromatic scale. The actual clustering, in this case was $\{A0, A6, B6\}$, $\{A2, A9\}$, $\{A7, B9\}$, $\{B3, B4\}$ and $\{C0, C2, C7\}$. Like Musician 1's clustering, there are significant differences from but also much structure in common with the "ground truth".

Cluster quality

Table 2 summarizes the results of evaluating cluster quality for all three experiments using the four clustering metrics (purity, Rand index, F-measure and NMI). First, note the high quality of the computational metrics—all of the 25 clusterings are very similar (or identical) to the clusterings induced by the three fitness functions (musical styles).

In comparison, note that, not surprisingly, the quality of the five clusterings resulting from the survey of non-musicians was lower and contained more variability. As mentioned earlier, reasons for this likely include the respondents' general lack of formal musical training, the inherent difficultly and subjectivity of the task and the relatively low number of responses and resulting variance in the generated distance matrix elements.

Finally, note the relatively better quality of the clusterings generated by the analyses of the trained musicians when compared with those generated by the survey of non-musicians. Formal musical training and familiarity with this type of task do result in measurable gains in cluster quality, suggesting that what the system produces can be, in fact, recognizable as distinct musical styles (at least to trained musicians). The musician-generated clusterings do show greater variability than the computational clusterings; nevertheless, the results are reasonable, with Musician 2 generating particularly high-quality clusters, comparable with the computational models.

Conclusion

We have demonstrated a system that employs a linear *multi-objective fitness function* for the generation of disparate musical styles. We have also presented several different computable musical features that can be incorporated in this multi-objective fitness function and thus form the basis of a musical style description. Three qualitatively different evaluation methods all confirm that the system can produce distinct and recognizable styles of music.

While any length of output is possible with our system, the ability to produce music which would engage an audience through a significant composition is lacking. The current system does not incorporate any global structure, and techniques that do so, allowing such things as the development of a musical theme and the utilization of contrasting themes would offer significant improvements in the scores produced.

A major element of music that was omitted from this research is rhythm. Every melody produced by the system has the same mono-rhythmic element and each feature extractor ignores the location in the beat structure where notes fall. Time signature, note placement in a measure and beat all impact the importance and perception of a note, and the system must be expanded to incorporate these rhythmic elements.

Yet another musical dimension ignored here is the addition of harmonization and polyphony. The addition of these complexifying musical elements is non-trivial and will require significant additional research both in how to properly incorporate the additional features and in how to mitigate the computational complexity that will likely result.

Such musical improvements, as well as input from musical experts (as in our third evaluation) will necessitate the development of additional feature extractors, e.g. for measuring rhythmic and polyphonic features as well as global structures, etc.

Finally, and perhaps most significantly, during the course of this work, *we* determined the parameters for the feature extractors, removing much of the creative responsibility from the system. An important next step is to add an element of autonomy to the system such that style of composition is chosen automatically using a meta-level fitness function. One possible step in this direction would be the automatic learning of feature extractor parameters from musical corpora. As an ancillary result, this could produce some new insights into parameterizing various known musical genres. However, even successful corpora-based parameter learning is still dependent on the (intelligent) selection of the corpus, suggesting, perhaps, another way to consider the problem of meta-level fitness.

References

Anders, T., and Miranda, E. R. 2009. A computational model that generalises Schoenberg's guidelines for favourable chord progressions. In *Proceedings of the Sound and Music Computing Conference*.

Biles, J.; Anderson, P.; and Loggi, L. 1996. Neural network fitness functions for a musical IGA. In *Proceedings of the International ICSC Symposium on Intelligent Industrial Automation and Soft Computing*, B39–B44. ICSC Academic Press.

Biles, J. 1994. Genjam: A genetic algorithm for generating jazz solos. In *Proceedings of the International Computer Music Conference*, 131–137.

Biles, J. 2001. Autonomous GenJam : Eliminating the fitness bottleneck by eliminating fitness. In *Proceedings of the GECCO-2001 Workshop on Non-routine Design with Evolutionary Systems*.

Brown, A. R. 2004. An aesthetic comparison of rule-based and genetic algorithms for generating melodies. *Organized Sound* 9(2):193–199.

Burton, A. R., and Vladimirova, T. 1999. Generation of musical sequences with genetic techniques. *Computer Music Journal* 23(4):pp. 59–73.

Cope, D. 2004. *Virtual Music: Computer Synthesis of Musical Style*. The MIT Press.

Deb, K., and Kalyanmoy, D. 2001. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 1st edition.

Freitas, A., and Guimarães, F. 2011. Melody harmonization in evolutionary music using multiobjective genetic algorithms. In *Proceedings of the Sound and Music Computing Conference*.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1st edition.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, Michigan: The University of Michigan Press.

Table 2: Quality of clustering results. The first two columns give results for the two different computational agglomerative clustering outcomes: the case where two songs are placed in the wrong cluster (19/25) and the case where all songs are assigned to the correct cluster (6/25); the next five columns give results for the five different agglomerative clustering techniques using the results of the survey of non-musicians as a distance measure; the last four columns report results from clusterings made by three members of the Faculty of Music (the first faculty member's responses could be taken two different ways, shown in two separate columns, M1a and M1b).

	Computational		Non-musician Survey					Musician Analysis			
	Miss 2	Perfect	Single	Complete	UnWtAvg	WtAvg	JBW	M1a	M1b	M2	M3
Purity	0.93	1.00	0.43	0.73	0.60	0.50	0.53	0.67	0.83	0.92	0.83
Rand Index	0.91	1.00	0.36	0.65	0.56	0.61	0.58	0.66	0.74	0.87	0.75
F-Measure	0.87	1.00	0.45	0.33	0.45	0.43	0.35	0.43	0.53	0.80	0.43
NMI	0.84	1.00	0.15	0.23	0.18	0.13	0.34	0.42	0.56	0.81	0.60

Johanson, B. E., and Poli, R. 1998. 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.

McIntyre, R. 1994. 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*, 852–857 vol.2.

Mckay, C., and Fujinaga, I. 2006. jSymbolic: A feature extractor for MIDI files. In *Proceedings of the International Computer Music Conference*, 302–305.

McKay, C. 2004. *Automatic genre classification of MIDI recordings*. Ph.D. Dissertation, McGill University.

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

Phon-amnuaisuk, S.; Tuson, A.; and Wiggins, G. 1999. Evolving musical harmonisation. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*.

Tanaka, T.; Nishimoto, T.; Ono, N.; and Sagayama, S. 2010. Automatic music composition based on counterpoint and imitation using stochastic models. In *Proceedings of the Sound and Music Computing Conference*.

Yip, C. L., and Kao, B. 1999. A study of musical features for melody databases. In Bench-Capon, T. J. M.; Soda, G.; and Tjoa, A. M., eds., *Database and Expert Systems Applications, 10th International Conference, DEXA '99, Florence, Italy, August 30 - September 3, 1999, Proceedings*, volume 1677 of *Lecture Notes in Computer Science*, 724–733. Springer.