NTNU
Norwegian University of
Science and Technology

Evolutionary Music Composition

A Quantitative Approach

Johannes Høydahl Jensen

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Pauline Haddow, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Artificial Evolution has shown great potential for musical tasks. However, a major challenge faced in Evolutionary Music Composition systems is finding a suitable fitness function. The aim of this research is to devise an *automatic* fitness function for the evolution of novel melodies.

Assignment given: 15. January 2011

Supervisor: Pauline Haddow, IDI

Abstract

Artificial Evolution has shown great potential in the musical domain. One task in which Evolutionary techniques have shown special promise is in the automatic creation or composition of music. However, a major challenge faced when constructing evolutionary music composition systems is finding a suitable fitness function.

Several approaches to fitness have been tried. The most common is interactive evaluation. However, major efficiency challenges with such an approach have inspired the search for *automatic* alternatives.

In this thesis, a music composition system is presented for the evolution of novel melodies. Motivated by the repetitive nature of music, a *quantitative* approach to automatic fitness is pursued. Two techniques are explored that both operate on frequency distributions of musical events. The first builds on *Zipf's Law*, which captures the scaling properties of music. Statistical *similarity* governs the second fitness function and incorporates additional domain knowledge learned from existing music pieces.

Promising results show that pleasant melodies can emerge through the application of these techniques. The melodies are found to exhibit several favourable musical properties, including rhythm, melodic locality and motifs.

Preface

This master's thesis presents the research completed during my final semester at the Norwegian University of Science and Technology (NTNU). It was carried out in the period January to June 2011 at the Department of Computer and Information Science (IDI). It is a continuation of my project work done as part of the preceding 2010 semester, which resulted in a paper submitted to the Genetic and Evolutionary Computation Conference 2011.

I would like to thank my supervisor, professor Pauline Haddow, for invaluable feedback and support. Our meetings were stimulating and always left me in an uplifted state.

Bill Manaris deserves my gratitude for his help and advice. Credit is given to Classical Archives (`www.classicalarchives.com`) for kindly providing access to their collection of music files.

I would also like to thank my fellow students at the CRAB lab for their friendship, advice and joyful social gatherings.

Thanks go to my family for their support and especially my father for his valuable feedback. Finally, I wish to thank Mari for her help and constant encouragement.

Johannes H. Jensen

Trondheim, 2011

Contents

Chapter 1

Introduction

> *If a composer could say what he had to say in words he would not bother trying to say it in music.* (Gustav Mahler)

Artificial Intelligence concerns the creation of computer systems that perform tasks that are normally addressed by humans. Music Composition is one such area. The task of creating music which human listeners would appreciate has been shown to be difficult with current AI techniques (Miranda and Biles, 2007). Although music exhibits many rational properties, it distinguishes itself by also involving human emotions and aesthetics, which are domains not fully understood and also difficult to describe mathematically.

What thought processes are followed when creating music? Asking an experienced composer is unlikely to yield a precise answer. The response might contain some general rules of thumb: a melody should encompass a few central motifs; the tones should fit the chord progression and so on. However, there are few formal rules that explain the process of music composition. The difficulty in formalizing music knowledge and understanding the process of music creation, makes music a challenging domain for computers (Minsky and Laske, 1992).

Evolutionary techniques have shown to be powerful for searching in complex domains too difficult to tackle using analytical methods (Floreano and Mattiussi, 2008). It is thus perhaps not so surprising that Artificial Evolution has shown potential in the musical domain. The most popular task pursued is *Evolutionary Music Composition* (EMC), where the goal is to create music through evolutionary techniques. However, a major challenge with such systems is the design of a suitable fitness function.

Some researchers have considered interactive fitness functions, i.e. fitness assigned by humans. However, interactive evaluation brings many efficiency problems and concerns (see Section 2.6.1). An *automatic* fitness function would steer evolution towards *pleasant* music, without extensive human input.

The term "pleasant" is chosen here instead of the subjective term "liking" because pleasantness in contrast to preference, shows little variation among subjects with different musical tastes (Ritossa and Rickard, 2004). In other words, pleasantness is in essence more of an invariant measurement than liking. For example, many people will admit that they find classical music *pleasant*, but certainly not all will claim to *like* classical music.

It is however difficult, if not impossible, to be objective when discussing music. There are simply too many factors that influence our musical perception. Thus when music is described as "pleasant" in this thesis, it is done so in an attempt to be as objective as possible.

1.1 Goals and Limitations

As stated in the problem description, the focus of this research is automatic fitness functions for the evolution of music. In particular, a fitness function is sought that can:

1. **Capture *pleasantness* in music**
2. **Be applied to evolve music in different *styles***

An automatic fitness able to estimate the pleasantness of music would make it possible to evolve music which is at least *consistently* appealing. In other words, it may serve as a baseline for evolution of music with more sophisticated properties.

Significant variation exists within the music domain. The ability to model different musical styles would allow similar diversity in evolved music as well. Assuming personal music taste is somewhat correlated to musical style, the capability to evolve similar properties in music is a clear advantage.

Finally, in order to reduce the complexity and narrow the scope of this work, the focus is restricted to evolution of short, *monophonic* melodies, i.e one note played at a time.

1.2 Overview of This Document

The thesis is structured as follows. Chapter 1 gives an introduction to the research domain, the motivation behind evolutionary music composition systems and the main problem areas explored by this research. In Chapter 2, important background material used throughout the thesis is covered. Music representation is discussed in detail in Section 2.5. An overview of the approaches to musical fitness functions, including a discussion of their strengths and weaknesses, is given in Section 2.6.

In Chapter 3, the theory and concepts employed in this research are presented. The music representation employed is described in Section 3.1, followed by the two approaches to fitness in Sections 3.3 and 3.4.

Chapters 4 and 5 presents the experiments performed with fitness based on Zipf's Law and distribution similarity, respectively. Conclusions from the experiments are drawn at the end of each chapter.

Finally, Chapter 6 concludes the research and suggests areas of future work.

Many music examples are given throughout the thesis, which are presented as printable music scores. The accompanying Zip-archive contains both MIDI and audio renderings of all the music examples.

Chapter 2

Background

This chapter covers important background material used throughout the rest of the thesis. Section 2.1 gives a short introduction to basic music terminology. Section 2.2 explains the main concepts behind Artificial Evolution. In Section 2.3, a short overview of Evolutionary Computation applied to aesthetic domains is given. Section 2.4 gives a survey of related work in the Evolutionary Music field.

Music representation schemes are covered in Section 2.5. Finally in Section 2.6, approaches to music fitness are discussed.

2.1 Music Terminology

This section gives a short introduction to the basic music terminology used throughout this document. If the reader is familiar with elementary music terminology, the section can safely be skipped.

Music comes in many forms and flavours, and the styles and conventions vary greatly across cultures. In *Western* music however, the common depiction of music is that of a *score*. A music score represents music as sequences of *notes, rests*, and other relevant information like tempo, time signature, key and lyrics. Figure 2.1 shows an example music score, which has a tempo of 120 beats per minute (BPM).

Notes and rests are the primary building blocks in the music domain, much as letters are the elementary units in language. The note value determines

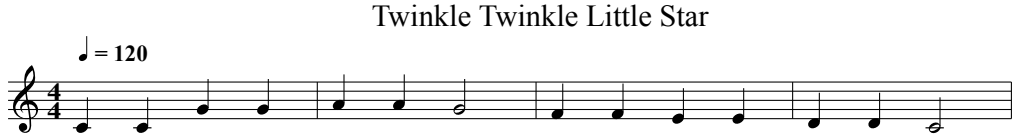
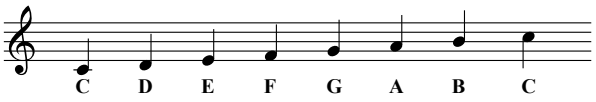


Figure 2.1 Example music score with 4 bars, 4 beats per bar and a tempo of 120 beats per minute.

(a) Five common note values. From the left: the whole note, half note, quarter note, eight note and sixteenth note.

(b) Two note modifiers: An augmented quarter note (left) and a quarter note tied together with an eight note (right).



(c) Note pitches ranging from Middle C to C_5, i.e. a one-octave C major scale.

Figure 2.2 Note values (a), note modifiers (b) and note pitches (c).

the *duration* of the note, while its vertical position in the score dictates the *pitch* that should be played. Equivalently, the rest value denotes the length of a rest (period of silence).

Note (rest) durations are valued as fractions of the whole note which typically lasts 4 beats. The half note lasts 1/2 of the whole note, the quarter note is 1/4th of the whole note and so on. The shape of the note (rest) determines its value (duration). Figure 2.2a shows the five most common note values.

Notes can also be *augmented*, denoted by a dot after the note. An augmented note lasts 1.5 times its original note value. Two notes of the same pitch may also be *tied* together, which indicates that they should be played as a single note, but for the duration of both. These two note *modifiers* are shown in Figure 2.2b. The resulting note value (duration) in both examples is the same: $\frac{1.5}{4} = \frac{1}{4} + \frac{1}{8} = \frac{3}{8}$.

Pitches are discretized into finite frequencies, corresponding to the keys on a piano. Pitches are grouped into 12 classes ranging from C to B. The *accidentals* modify a pitch: the sharp ♯ raises the pitch by a semitone (half-step) and the flat ♭ lowers it by the same amount. The interval between two

pitches of the same class is called an *octave*, where the upper pitch has a frequency that is twice that of the lower pitch. Figure 2.2c shows how the different pitches are organized in a music score.

Musical phrases commonly follow a *scale* which denotes which pitches are to be present. For instance, the *C-major* scale consists of all the white keys on the piano, while the *chromatic* scale contains all keys (both white and black).

A music score divides time into discrete time intervals called *bars* (measures) which are separated by a vertical line. The *time signature* appears at the start of the score: the upper number specifies how many *beats* are in a bar and the lower number determines which note value forms a beat. For instance, a time signature of $\frac{3}{4}$ (the waltz) means that there are 3 beats in a bar and the quarter note constitutes a beat. The score in Figure 2.1 consists of 4 bars and has a time signature of $\frac{4}{4}$.

Looser musical structures are also useful when describing music. *Motifs* are short musical ideas, i.e. short sequences of notes which are repeated and unfolded within a melody. Several motifs can be combined to form a *phrase*. A *theme* is the central material, usually a melody, on which the music is founded.

2.2 Evolutionary Computation

Evolutionary Computation (EC) incorporates a wide set of algorithms that take inspiration from natural evolution and genetics. Applications include solving hard optimization problems, design of digital circuits, creation of novel computer programs and several other areas that are typically addressed by human design. Most evolutionary algorithms are based on the same key elements found in natural evolution: a *population*, *diversity* within the population, a *selection mechanism* and *genetic inheritance* (Floreano and Mattiussi, 2008).

In an Evolutionary Algorithm (EA) a population of *individuals* is maintained that represent possible *solutions* to a target problem. Each solution is encoded as a *genome* which, when decoded, yields the candidate solution. The structure of the genomes is called its *genotype*, and a wide range of possible representations exist. The *phenotype* describes the form of the individual and

is mostly problem specific. While the phenotype directly describes a solution, the genotype typically employs a more low-level and compact encoding scheme.

An EA typically operates on *generations* of populations. For each evolutionary step, a selection of individuals are chosen for *survival* to the next generation. The selection mechanism typically favours individuals according to how well they solve the target problem; that is, according to their *fitness*.

Tournament selection is a popular selection mechanism, in which several "tournaments" are held between k randomly chosen individuals. With probability $1 - e$, the individual with the highest fitness wins the tournament. With probability e, however, a random individual is chosen instead. This mechanism promotes genetic diversity while at the same time maintaining selection pressure.

The *fitness function* takes as input an individual which is evaluated to produce a numerical fitness score. As such it is highly problem specific and often involves multiple objectives. A common technique is to combine several objectives as a weighted sum.

In order to explore the solution space, diversity is maintained within the population through genetic operators like *mutation* and *crossover*. Random mutations are introduced to the population in order to explore the close neighbourhood of each solution, in search for a local optima. That is, mutations cause *exploitation* of the best solutions found so far.

Many EAs also include a second selection phase called *reproduction*. During reproduction, two (or more) parent individuals are chosen for mating, and their genetic material is combined to produce one or more offspring. Each offspring then *inherits* genetic material from its parents. The idea is that combining two good solutions might result in an even better solution. The combination of different genetic material is addressed by the *crossover* operator, to which many variations exist. Crossover ensures *exploration* of the solution space.

The evolutionary loop continues like this with selection, (reproduction) and mutation through many generations until some stopping criterion is met, and a satisfying solution emerges from the population. Figure 2.3 depicts the evolutionary loop and its main components.

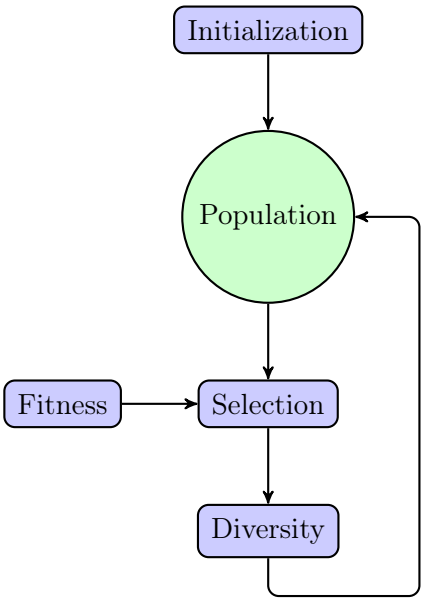


Figure 2.3 A high level diagram of the evolutionary loop.

In general, the process of constructing an EA involves:

1. Design of a *genetic representation* (genotype) and its corresponding *genetic operators* (mutation, crossover, etc.)

2. Creation of a *mapping* between the genotype to the phenotype

3. Choosing the *selection mechanisms*

4. Design of a *fitness function* which measures the quality of each solution

Different types of EAs mainly employ different genetic representations and operators. Genetic Algorithms (GA) operate on *binary* representations, Genetic Programming (GP) uses a *tree-based* scheme and Evolutionary Programming (EP) operates directly on phenotype parameters. For a detailed introduction to Evolutionary Computation, see Floreano and Mattiussi (2008).

2.3 Evolutionary Art and Aesthetics

Music is an art form with and thus exhibits an important aesthetic aspect. This section gives a brief overview of how evolutionary computation has been applied in other aesthetic domains.

The first to utilize evolutionary techniques in the aesthetic field was Richard Dawkins, the evolutionary biologist of later renown, who in 1987 devised "*Biomorphs*", a program that let the user evolve graphical stick figures. Since then, a myriad of evolutionary art systems have been developed (Bentley and Corne, 2002).

Another early pioneer in the field was Karl Sims, who applied interactive evolution for computer graphics. Both 3D and 2D images as well as animations were explored, using a symbolic Lisp tree representation. Completely unexpected images could emerge, some remarkably beautiful (Sims, 1991). He presented his work at Paris Centres Pompidou in 1993. Named *Genetic Images*, the artwork allowed museum visitors to serve as selectors for new generations of evolved images.

Another similar example is NEvAr by Machado and Cardoso (2002), a system which lets the user evolve colour images using Genetic Programming and user-provided fitness assessment.

Secretan et al. (2008) take the concept a step further with *Picbreeder* – a web-based service where users can evolve pictures collaboratively. It offers an online community where pictures can be shared and evolved further into new designs.

Common to these examples is the interactive approach to fitness, where humans pass aesthetic judgement on the generated pieces. Since humans act as selectors for future generations, these systems are good demonstrations of the power of evolutionary techniques in the aesthetic domains.

Evolution has also been applied in other areas where aesthetics play an important role. Architecture, sculptures, ship design and sound are just a few examples. For a thorough overview of evolutionary art, see Bentley and Corne (2002).

2.4 Evolutionary Music

One of the early examples of Evolutionary Computation applied for music is Horner and Goldberg (1991), where a Genetic Algorithm was used to perform thematic bridging. Since then, EC has been used in a wide range of musical tasks, including sound synthesis, improvisation, expressive performance analysis and *music composition*.

Evolutionary Music Composition (EMC) is the application of Evolutionary Computation for the task of creating (generating) music. In EMC systems, the genotype is typically binary (GA) or tree-based (GP). The phenotype is the music score – sequences of notes and rests that make up the composition.

As mentioned earlier, the fitness function poses a significant challenge when constructing evolutionary composition systems. Approaches include interactive evaluation, hardwired rules and machine learning methods.

One well-known EMC system is *GenJam*, a genetic algorithm for generating jazz solos (Biles, 1994). A binary genotype is employed, with musical (domain specific) genetic operators. GenJam uses interactive fitness evaluation, where a human musical mentor listens through and rates the evolved music phrases.

Johanson and Poli (1998) developed a similar EMC system called *GP-Music* which can evolve short musical sequences. Like GenJam, it relies on interactive fitness evaluation, but employs a tree-based genotype instead of a bit string.

Motivated by subjectivity and efficiency concerns with interactive fitness assessment, Papadopoulos and Wiggins (1998) developed an *automatic* fitness function based on music-theoretical rules. It was applied in a GA for the evolution of jazz melodies, with some promising results.

Another similar example is *AMUSE*, where a hardwired fitness function is used to evolve melodies in given scales and chord progressions (Özcan and Erçal, 2008).

Phon-Amnuaisuk et al. (1999) used a GA to generate traditional musical harmony, i.e. chord progressions. A fitness function derived from music theory was used and surprisingly successful results are reported.

The Swedish composer Palle Dahlstedt presents an evolutionary system which can create complete piano pieces. By using a recursive binary tree representation, combined with formalized fitness criteria, convincing contemporary performances have been generated (Dahlstedt, 2007).

Some have considered probabilistic inference as framework for automatic fitness functions. A recent example is *Little Ludwig* (Bellinger, 2011), which learns to compose in the style of known music. Evolved music is evaluated based on the probability of note sequences with respect to some inspirational music piece.

This has been a short introduction to the EMC field and is by no means exhaustive. For a more in-depth survey, the reader is referred to a book dedicated to the subject – "Evolutionary Computer Music" (Miranda and Biles, 2007).

2.5 Music Representation

The genotypes most commonly used in EMC systems are *binary* bit strings (GA) and *trees* (GP). Although the choice of genotype differs, music domain knowledge is commonly encoded into the representation in order to constrain the search space and get more musical results.

Discrete MIDI[1] pitches are usually employed instead of pitch frequencies. They range from 0-127 and include all the note pitches possible in a music score.

2.5.1 Linear Representations

Linear genotypes encode music as a sequence of notes, rests and other musical events in an array-like structure. Binary genotypes fall under this category, where the sequence is encoded in a string of bits. Symbolic vectors are also common, which are at a higher level of abstraction and somewhat easier to deal with. Linear genotypes fall under the Genetic Algorithm (GA) class of EAs.

The genotype is typically of a fixed size specified by the user, dividing the genotype into N slots which each can hold a musical event. This effectively limits the length of the evolved music, either directly or implicitly. A

[1]MIDI (Musical Instrument Digital Interface) is a standardized protocol developed by the music industry which enables electronic instruments to communicate with each other. For more information see the website `http://www.midi.org/`.

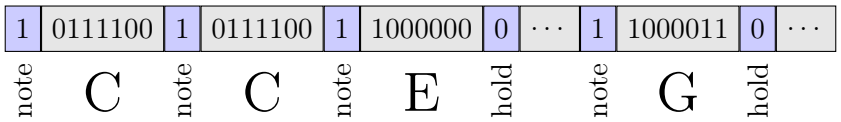


Figure 2.4 Binary genome with implicit note durations. Each event is one byte long. The first bit denotes the event type, either *note* (1) or *hold* (0). The remaining seven bits contain the MIDI pitch to be played for the note event and are simply ignored in the case of a hold event.

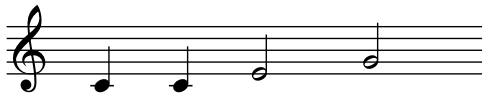


Figure 2.5 Music score corresponding to the genome in Figure 2.4, given that event durations are one quarter note long.

position-based scheme is usually employed, i.e. a note's position in the score is implicitly determined by its location in the genome.

Musical events can include notes, rests, chords, dynamics (e.g. loud or soft), articulation (e.g. staccato) and so on. The *duration* of an event is either specified explicitly in the event, or determined implicitly, e.g. by the events that follow.

Papadopoulos and Wiggins (1998) use a symbolic vector genotype, where the duration of each event is explicitly defined. Rather than absolute pitches, a degree-based scheme is used where the pitches are mapped to degrees of a pre-defined scale. Genomes are thus sequences of (degree, duration) tuples.

An implicit duration scheme is found in GenJam's binary genotype. Three types of events are supported: *note*, *rest* and *hold*. A *note* specifies the start of a new note. Similarly a *rest* event dictates the beginning of a rest. Both events implicitly mark the end of any previous note or rest. Finally, a *hold* event results in the lengthening of a preceding note (rest) by a fixed amount (Biles, 1994).

A binary genome with an implicit duration scheme can be seen in Figure 2.4. Each event is one byte long – one bit for the event type and seven bits for the MIDI pitch to be played. In the case of a hold event, the pitch field is simply ignored. Figure 2.5 shows the corresponding music score, given that event durations are one quarter note long.

Music knowledge is often embedded into the genetic operators. For example, instead of the random point mutation traditionally employed in GAs, various musical permutations can be used. Examples include reversing, rotating and transposing random sequences of notes, as well as copying a fragment from one location to another.

Finally, linear genotypes are simple to use, since their structure closely resembles a music score. Interpretation of the genome is thus fairly straightforward for a human.

2.5.2 Tree-Based Representations

Tree-based genotypes encode music as a tree where the leaf nodes (the *terminals*) contain the notes and rests, and the inner nodes represent operators that perform some *function* on the contents of their sub-trees. The tree is recursively parsed to produce a music score. The functions must minimally include some form of *concatenation* to be able to produce sequences of notes, but often other musical operators are used as well. Tree-based genotypes are part of the Genetic Programming (GP) family of EAs.

Some argue that GP is well suited for music because the tree closely resembles the *hierarchical structure* found in music, e.g. short sequences of notes make a motif, several motifs form a phrase which combines into melodies and larger sections (Minsky and Laske, 1992).

Johanson and Poli (1998) employ seven different operators (functions), including concatenation, repetition, note elongation, mirroring and transposition of notes. Manaris et al. (2007) also include functions for polyphony, retrograde and inversion.

Figure 2.6 depicts an example tree-based genome for music, with three different operators. The leaf nodes are (pitch, duration) pairs. The *repeat* operator causes the notes in its sub-tree to be played twice, while *slow* doubles their duration. The corresponding music score can be seen in Figure 2.5.

As can be seen, music knowledge is embedded into the function nodes. As such the complexity lies in the interpretation of the genome and the mapping to a music score, i.e. in the phenotype, at a higher level of abstraction than in the linear genotypes.

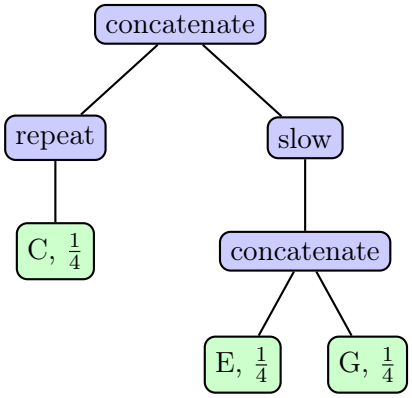


Figure 2.6 An example tree-based music genome. The leaves contain (pitch, duration) pairs and the non-leaf operators perform concatenation, repetition or slowing of the notes in their respective sub-trees. The corresponding music score can be seen in Figure 2.5.

Since the domain knowledge lies at the phenotype level (when parsing the tree), traditional genetic operators from GP can be used without further modification. Mutation simply replaces a random node with a randomly generated sub-tree, while crossover swaps two randomly selected sub-trees between genomes. Care must be taken when performing these operations so that the tree doesn't grow out of bounds, resulting in extremely long pieces of music. Typically a maximum depth parameter is implemented to limit the height of the tree.

Tree-based genotypes are dynamic in size, meaning that the length of the gnomes can change across generations. This in turn allows variation in the length of the evolved music. Introducing new music knowledge or adding more musical possibilities (like polyphony or dynamics) is also fairly straightforward, and simply involves implementation of a new function node. The structure of the genotype remains the same.

It should be noted that trees are more complex structures than vectors and are therefore more difficult to use. For a human, translation of a tree into a music score is not exactly child's play.

2.5.3 Phenotype Mapping

The translation from genotype to phenotype often adds further music knowledge to the system, often in the form of musical constraints.

One common technique is to map the pitches in the genotype to a pre-defined *scale*. The scale can either be kept the same for the entire score, or follow some chord progression specified by the user.

GenJam makes use of this approach, where jazz melodies are mapped to scales according to a predefined chord progression. This allows the melodies to be applied to many different musical contexts. Since the notes are always in scale, the system never plays a "wrong" note. This scheme effectively reduces the amount of unpleasant melodies the mentor has to listen through (Biles, 1994). In the context of EAs, the search space is reduced to melodies in certain scales.

In many musical genres, however, breaking the "rules" from time to time is encouraged and can give rise to much more interesting music. A jazz soloist who always keeps to the "correct" scales usually results in a rather boring performance.

Another feature often employed is that of a *reference pitch* which specifies the lowest possible pitch to consider during the mapping process. A reference duration can also be considered, to effectively adjust the tempo of the resulting score (given a fixed BPM).

2.6 Fitness

As mentioned, musical fitness is the primary focus of this research and has proven to be the most challenging aspect of EMC systems. Some measurement of the "goodness" of a music piece is required in order to guide evolution towards aesthetically pleasing individuals. In other words, a fitness function is needed which captures the human perception of pleasantness in music.

Unfortunately, as discussed in Chapter 1, human music creation is an elusive process, and exactly what constitutes pleasant music is also not fully understood. Furthermore, personal musical taste plays an important role in the liking of a given piece of music. Certain styles of music will appeal to some people, while not to others. A fitness function which can be tuned to favour certain musical styles would certainly be beneficial, allowing the generation of a wider spectrum of music.

The approaches to musical fitness can roughly be divided into three categories:

- *Interactive evaluation*: fitness assigned by humans, e.g. Biles, 1994.
- *Hardwired fitness functions*: rules based on music theory or experience, e.g. Papadopoulos and Wiggins, 1998; Dahlstedt, 2007.
- *Learned fitness functions*: Artificial Neural Networks (ANNs), Machine Learning, Markov Models, e.g. Bellinger, 2011.

A more detailed explanation of the different approaches to fitness follows, with a discussion of their main strengths and weaknesses.

2.6.1 Interactive Evaluation

Probably the most precise fitness assessment available is from the target audiences themselves. If humans evaluate the fitness of the music pieces, the problem of *formalizing* music knowledge is avoided. Surely this produces a much more accurate assessment of the "goodness" of a music piece than any known algorithmic method?

With interactive fitness functions, one or more human mentors will sit down and listen through each evolved music piece and give them a fitness score, either explicitly or implicitly.

In GenJam, for example, the mentor will press 'G' on his keyboard a number of times to indicate that he likes what was just played. Similarly, to indicate that a portion of the music was bad, he presses 'B'. Fitness is then simply the number of Gs minus the number of Bs (Biles, 1994).

One of the main problems with the interactive approach is that for every music piece in the population, the mentors must listen carefully through each one and determine which ones are better. This process has to be repeated for each generation.

In the visual art domains, this is less problematic because several images can be presented to the mentor at the same time and thus be compared quicker and easier. Music, however, is *temporal* in nature. A piece of music always has to be presented in the correct tempo and the length of the piece dictates

the minimum time required for evaluation. Finally, several music pieces can not be presented simultaneously, making the evaluation process much more time consuming than with images.

This issue has been termed "the fitness bottleneck" and is the main limiting factor for the population size and number of generations (Biles, 1994). Interactive fitness might provide a good evaluation, but clearly at the expense of *efficiency*.

Another important problem with the interactive approach is *subjectivity*. The humans responsible for evaluating each music piece will most likely be biased towards their own musical taste. This can be countered by increasing the number of mentors to gain statistical significance. However, much care must be taken when selecting the participants. The number of people, their age, background etc. are parameters which will clearly affect the results. Determining the important parameters and gathering the right people is a challenging task in itself.

Furthermore, providing *consistent* evaluation is hard and it is likely that the mentors will be biased from previous listenings, mood or even boredom. Finally, interactive fitness tells us very little about the processes involved in music composition. The music knowledge which is applied is hidden away in the mentor's mind and is for that reason of limited research value (Papadopoulos and Wiggins, 1998).

2.6.2 Hardwired Fitness Functions

Another approach to musical fitness is to study music theory (or other sources) for best practices in music composition. This typically involves examination of relevant music theory and translation of this knowledge to algorithmic fitness assessments. Such methods attempt to address the challenges found in interactive fitness functions.

A fitness function for melodies based on jazz theory was designed by Papadopoulos and Wiggins (1998). The function consists of a weighted sum of eight sub-objectives related to melodic intervals, pattern matching, rhythm, contour and speed. The authors report that their system generated subjectively interesting melodies, although few examples are provided. Noticeably the more music knowledge which was encoded into the system, the better were the results.

A similar rule-based approach is found in Özcan and Erçal (2008). A survey performed with 36 students revealed that the subjects were unable to differentiate between the melodies created by the system and those of a human amateur musician. Further, tests showed a correlation between the fitness of a melody and its statistical rank as given by the human evaluators.

As discussed above, hardwired fitness functions can yield good results. However, they can often be quite challenging to design. The rules of thumb found in most music literature are often vague and hard to interpret algorithmically. They are only *best practices* and are certainly not followed rigidly by most artists. Furthermore, sufficient knowledge might not even be available in the literature for certain styles of music.

Another issue is *scalability*. Hardwired fitness functions tend to become highly specialized towards some small subset of music. In order to evolve music in another style, the rules that make up the fitness have to be altered and redesigned by hand – a potentially challenging and time consuming process.

2.6.3 Learned Fitness Functions

Because of the difficulty in hand-designing good fitness functions for music, many researchers have turned to *learned* fitness functions for possible solutions. In this approach, machine learning techniques are used to train the fitness function to evaluate music pieces. Such systems learn by extracting knowledge from examples (training data) which are utilized to evaluate new unseen music pieces.

One of the main advantages of machine learning approaches is that they typically require less domain knowledge than hardwired fitness functions. Furthermore, they might discover novel aspects of music which could otherwise be missed by human experts.

Another important advantage is that of *adaptability*. Hardwired fitness functions are challenging and time consuming to create, and the design process has to be repeated for each musical style. A learned fitness function would ideally only require a new set of training data for analysis.

Unfortunately, machine learning techniques are not powerful enough to process raw music directly. For instance, passing an entire music score as the

input to an ANN will not only require an infeasible number of neurons, but the network is unlikely to succeed in extracting any relevant knowledge.

Some form of *feature extraction* is necessary to both reduce the dimensionality of the problem and assist the algorithm by identifying potentially useful musical characteristics. Identifying features which are musically meaningful is crucial for the algorithm to successfully learn anything.

Relevant, unbiased training data is essential, as well as collecting the sufficient amount of material in order to achieve good performance (Duda et al., 2006). For instance, a collection of music pieces used as training data should include music in all relevant musical styles and by many different authors.

In an attempt to improve the efficiency of GenJam, an ANN was trained based on data gathered from interactive evaluation runs. The hope was that the neural network would learn how to evaluate new music. However, the results were unsuccessful, with diverging fitness for nearly identical genomes (Biles et al., 1996). A similar attempt was made in Johanson and Poli (1998) with inconclusive results – some of the evolved melodies were reportedly nice while others rather unpleasant.

Even though learned fitness functions have great potential, they are challenging to design and tune. As mentioned, the disadvantages include sensitivity to the input data, many input parameters and difficult feature selection.

Chapter 3

Methodology

This thesis proposes a *quantitative* approach to Evolutionary Music Composition. That is, the fitness function operates on the *frequency* of music *events*. The evolutionary system is designed to create short, monophonic melodies.

The term "event" is used broadly here, meaning an occurrence of any kind within a music piece. In other words, an event is not restricted to single notes, but can be the relationship between notes or pairs of notes, for example. The types of events explored are covered in Section 3.2.1: *Metrics*.

Two different techniques for processing the frequency distributions are investigated, i.e. two fitness functions. The first builds on *Zipf's Law*, which measures the scaling properties in music and is described in Section 3.3. The second technique is based on the *similarity* between frequency distributions and is presented in 3.4.

A *tree-based* representation is employed based on Genetic Programming (see Section 2.5.2). Many of the features described in Section 2.5.3 are also included. A detailed description of the genotype and phenotype is given in Section 3.1.

3.1 Music Representation

In previous work by the author (Jensen, 2010) a symbolic, vector-based genotype was used to evolve music. As discussed in Section 2.5.1, linear representations (e.g. binary, vectors) are commonly found in the EMC literature. They all fall under the Genetic Algorithm class of Evolutionary Computation.

The other type of representation commonly employed is the tree-based genotype which falls under the Genetic Programming umbrella. It has been argued that GP is more suitable because the tree closely resembles the hierarchical structure found in music (see Section 2.5.2).

3.1.1 Introduction

In this work, a tree-based genotype is employed, which was shown to outperform the vector-based genotypes previously used (see Section 4.1). As discussed earlier, music is encoded in a tree where the leaf nodes (*terminals*) contain notes and the inner nodes represent *functions* that perform some operations on their sub-trees. Recursively parsing a tree results in a sequence of notes – the music score (see Section 2.5.2).

The following sections give an in-depth description of the tree-based representation used throughout this research.

3.1.2 Parameters

The genotype and phenotype take several parameters which determine various aspects of the representation. They are summarized here and described in more detail in the sections that follow.

Genotype Parameters

Pitches: Number of pitches (integer)

Durations: Number of durations (integer)

Max-depth: Maximum tree depth (integer)

Initialization method: Tree generation method ("grow" or "full")

Function probability: Probability of function nodes (float)

Terminal probability: Probability of terminal nodes (float)

Phenotype Parameters

Pitch reference: Lowest possible pitch (integer)

Scale: A musical scale mapping (list)

Resolution: Base note duration (integer)

Duration map: Set of durations to use (list)

3.1.3 Functions and Terminals

Inner nodes have two children, i.e functions take two arguments, thus resulting in binary trees. The *function set* only contains one function, *concatenation*, which is denoted by a "+". Concatenation simply connects the notes from its two sub-trees to a form a longer sequence. Although it would be interesting to include other functions as well, it was decided to keep things simple so that the fitness function would be responsible for most of the music knowledge.

The set of terminals contain the notes, which are represented as (pitch, duration) tuples. Pitches and durations are both positive integers, which are in the range $[0, N)$ where N is dictated by the the *pitches* or *durations* parameters. The way the pitches and durations are interpreted depends on the phenotype parameters and is detailed in Section 3.1.6. Note that rests are not included in the representation.

3.1.4 Initialization

Initialization of random genome trees is performed in two different occasions:

1. When creating the initial population at the beginning of the EA.
2. When generating random sub-trees for mutation.

Trees are generated in a recursive manner, where in each step a random node from either the function set or the terminal set is chosen. Tree growth is constrained by the *max-depth* parameter, which determines the maximum height of the generated trees.

The *initialization method* determines how each node is chosen. With the *full* method, a function node is always chosen before the maximum depth is reached, after which only terminals can be selected. This results in full binary trees with 2^D leaf nodes (notes), where D is the maximum depth. With the *grow* method, function and terminal nodes are chosen randomly according to the *function-* and *terminal probability* parameters, respectively. The resulting trees are therefore of varying size and shape. See also Koza (1992).

3.1.5 Genetic Operators

The traditional GP operators of mutation and crossover are adopted. *Mutation* replaces a random node with a randomly generated sub-tree. *Crossover* swaps two random sub-trees between genomes. Both operators conform to the maximum depth, meaning that the resulting genomes will never be higher than this limit.

For both operators, function and terminal nodes are selected according to the *function-* and *terminal probability*, respectively. As suggested in Koza (1992), the default function probability is 90% and terminal probability 10%.

3.1.6 Parsing

Parsing of the tree is done at the phenotype level, where the tree is recursively traversed. The result is the sequence of notes that make up the music score.

Genotype pitches p in the terminal nodes are offset by the *pitch reference* and mapped to the specified musical *scale*:

$$pitch = pitch_{ref} + 12\,\lfloor p/N \rfloor + scale\,[p \bmod N]$$

where N is the length of the *scale* list. In words, the second term calculates the octave while the last term performs the scale mapping. The resulting integer is a MIDI pitch number.

The genotype durations d (positive integers) can be interpreted in two ways:

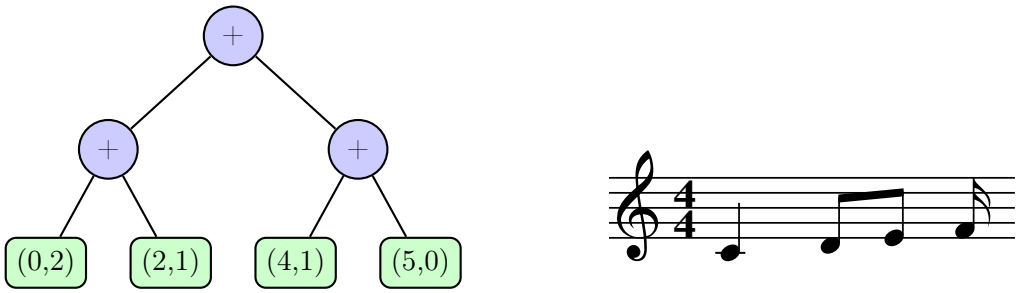


Figure 3.1 An example tree genome (a) and the resulting music score (b).

1. If the *resolution* parameter is specified, according to the equation: $\frac{2^d}{r}$, where r is the resolution. Thus for a resolution of 16, this would result in the real-valued durations $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$ for $d \in [0, 3]$.

2. If a *duration map* is given, d is interpreted as the index of an element in this map, i.e. $duration\,[d]$. This allows the use of durations that are not easily enumerated.

Figure 3.1 shows an example tree genome and the resulting music score after parsing, using a resolution of 16, pitch reference set to 60 (Middle C) and the chromatic scale (no scale).

3.2 Fitness

As mentioned, two approaches to automatic fitness for music are explored. The first is based on Zipf's Law, and is described further in Section 3.3. The second is based on distribution similarity and is detailed in Section 3.4.

Common to the two fitness functions is the *quantitative* approach – they operate on the *frequency* of musical events. The difference is how these frequency distributions are utilized to produce a fitness score. Figure 3.2 depicts the structure and relationship between the fitness functions. For both functions, the fitness score is calculated with respect to a set of *target* measurements.

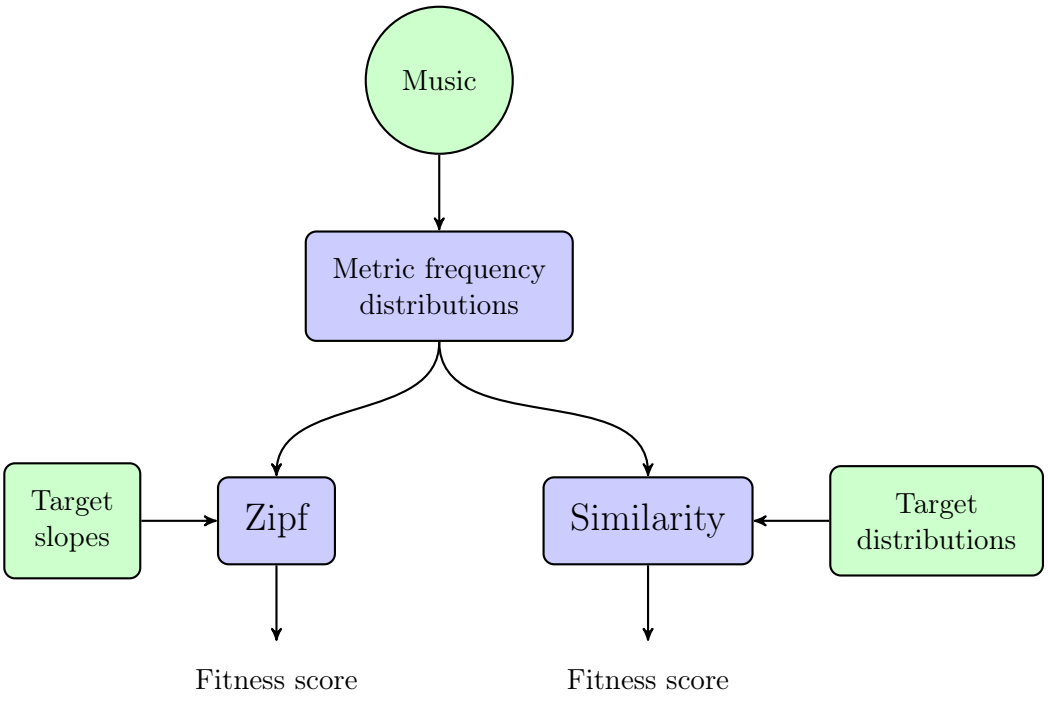


Figure 3.2 Diagram of the two fitness functions, the first based on Zipf's Law and the second on distribution similarity. The fitness score is calculated with respect to a set of *target* measurements.

3.2.1 Metrics

The musical events are produced by different *metrics* which are summarized here and described below. Most of the metrics are derived from Manaris et al. (2007).

Pitch: Note pitches ($p \in [0, 127]$)

Chromatic-tone: Note pitches modulo 12 ($ct \in [0, 11]$)

Duration: Note durations ($d \in \mathbb{R}^+$)

Pitch duration: Note pitch and duration pairs (p, d)

Chromatic-tone duration: Chromatic tone and duration pairs (ct, d)

Pitch distance: Time intervals between pitch repetitions ($t_p \in \mathbb{R}^+$)

Chromatic-tone distance: Time intervals between chromatic-tone repetitions ($t_{ct} \in \mathbb{R}^+$)

Melodic interval: Musical intervals within melody ($mi = p_i - p_{i-1}$ or absolute $mi = |p_i - p_{i-1}|$)

Melodic bigram: Pairs of adjacent melodic intervals (mi_i, mi_{i+1})

Melodic trigram: Triplets of adjacent melodic intervals $(mi_i, mi_{i+1}, mi_{i+2})$

Rhythm: Note durations plus subsequent rests ($r \in \mathbb{R}^+$)

Rhythmic interval: Relationship between adjacent note rhythms ($ri = \frac{r_i}{r_{i-1}}$)

Rhythmic bigram: Pairs of adjacent rhythmic intervals (ri_i, ri_{i+1})

Rhythmic trigram: Triplets of adjacent rhythmic intervals $(ri_i, ri_{i+1}, ri_{i+2})$

Pitches are positive integers corresponding to MIDI pitches, while durations are positive real numbers denoting a time interval. Chromatic tone is simply the octave-independent pitch, e.g. any C will have the chromatic tone number 0, Es will have the number 4 and so on. This is useful because pitches are perceptually invariant (perceived as the same) over octaves (Hulse et al., 1992).

Melodic intervals capture the distance between adjacent note pitches within the melody. As such they provide melodic information independent of the musical key, e.g. the same melody played in the C major and D major scale will contain the same melodic intervals. Hulse et al. (1992) presents evidence that melodies with the same sequence of intervals are perceptually invariant.

The melodic intervals come in two flavours: relative and absolute – the latter discards information about the direction of the interval. The melodic bigrams and trigrams produce pairs and triplets of melodic intervals, respectively.

The rhythm metric was created to describe rhythmic features formed by notes followed by any number of rests. When there are no rests, rhythm is equivalent to the duration metric. Since the genotype does not support rests, this is always the case for the evolved music. Rhythm is however useful when applied to real-world music which do contain rests.

Rhythmic intervals capture the relationship between two adjacent rhythms as a ratio. They are therefore independent of tempo and are the rhythmic equivalent of melodic intervals. Hulse et al. (1992) also demonstrates that rhythmic structure is perceptually invariant across tempo changes.

A rhythmic interval of 1.0 indicates no rhythmic change between two notes. An interval of less than 1.0 describes an increase in speed, while greater than 1.0 indicates a decrease. Rhythmic bigrams and trigrams are pairs and triplets of rhythmic intervals, respectively.

3.3 Fitness Based on Zipf's Law

Research on *Zipf's Law* has demonstrated that art tends to follow a balance between chaos and monotony. Evidence shows that this is also the case in music (Manaris et al., 2005, 2003, 2007). Building on this research, an automatic fitness function based on Zipf's Law is proposed for the evolution of novel melodies.

3.3.1 Zipf's Law

The Harvard linguist George Kingsley Zipf studied statistical occurrences in natural and social phenomena. He defined *Zipf's Law* which describes

the *scaling properties* of many of these phenomena (Zipf, 1949). The law states that the *"frequency of an event is inversely proportional to its statistical rank"*:

$$f = r^{-a} \tag{3.1}$$

where f is the frequency of occurrence of some event, r is its statistical rank and a is close to 1 (Manaris et al., 2005).

For example, ranking the different words in a book by their frequency, the most frequent word (rank 1) will occur approximately twice as often as the second most frequent word (rank 2), three times as often as the third most frequent word (rank 3) and so on. Plotting these ranks and frequencies on a logarithmic scale produces a straight line with a slope of -1. The slope corresponds to the exponent $-a$ in equation (3.1). Such a plot is known as a *rank-frequency* distribution.

Figure 3.3 shows the rank-frequency distribution of the 10,000 most frequent words in the *Brown Corpus* text collection and a straight line which fits the distribution. As predicted, the slope of the line is approximately -1.

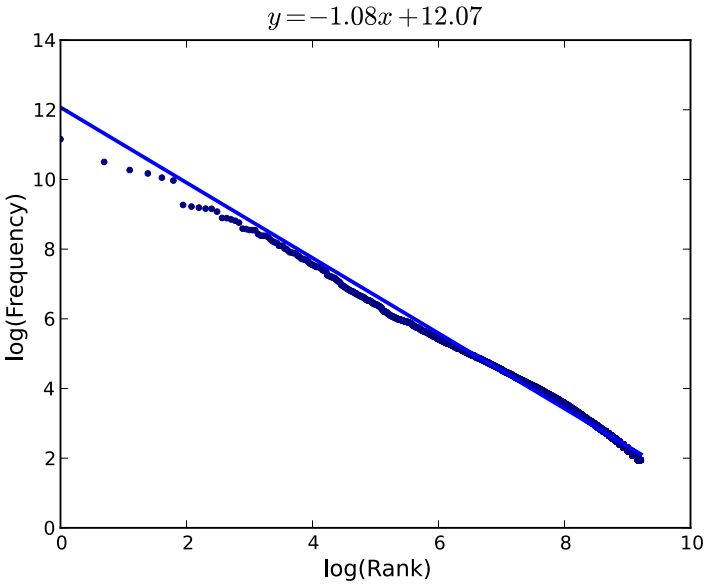


Figure 3.3 Rank-frequency distribution of the 10,000 most frequent words in the *Brown Corpus* and a straight line which fits the distribution. The slope of the line is approximately -1 as predicted by Zipf's Law.

Zipf's Law is a special case of a power law. When a is 1, the distribution is called $1/f$ noise, or *pink noise*. These $1/f$ distributions have been observed in a wide range of human and natural phenomena, including language, city sizes, incomes, earthquake magnitudes, extinctions of species and in various art forms. Other related distributions are *white noise* ($1/f^0$ – uniform random) and *brown noise* ($1/f^2$).

3.3.2 Zipf's Law in Music

In his seminal book *Human Behaviour and the Principle of Least Effort*, Zipf also found evidence for his theory in music. Analysis of Mozart's *Bassoon Concerto in Bb Major* revealed an inverse linear relationship between the length of intervals between repetitions of notes and their frequency of occurrence (Zipf, 1949).

Vossa and Clarke (1978) studied noise sources for stochastic music composition. They generated music using a white, pink and brown noise source. Samples of the results were played to several hundred people. They discovered that the music from the pink noise source was generally perceived as much more interesting than music from the white and brown noise sources.

Manaris et al. (2003) devised more metrics based on Zipf's Law, and later work expanded and refined them (Manaris et al., 2005, 2007). Each metric counts the frequency of some musical event and plots them against their statistical rank on a log-log scale. Linear regression is then performed on the data to estimate the *slope* of the distribution. The slopes may range from zero to negative infinity, indicating uniform random to monotone distributions, respectively. The coefficient of determination R^2 is also computed to see how well the slope fits the data. R^2 values range from 0.0 (worst fit) to 1.0 (perfect fit).

Some of the relevant metrics explored by Manaris include rank-frequency distributions of: pitches, chromatic tones, note durations, pitch durations, chromatic tone durations, pitch distances, melodic intervals, melodic bigrams and melodic trigrams. Section 3.2.1 covers the metrics in more detail.

Figure 3.4 shows the rank-frequency distributions and slopes from all the above metrics applied to *The Beatles' Let It Be*. Most of the metrics display slopes near -1 as predicted. Notice the rather steep slope of -2.0 for note durations, something which suggests little variation in the music rhythm.

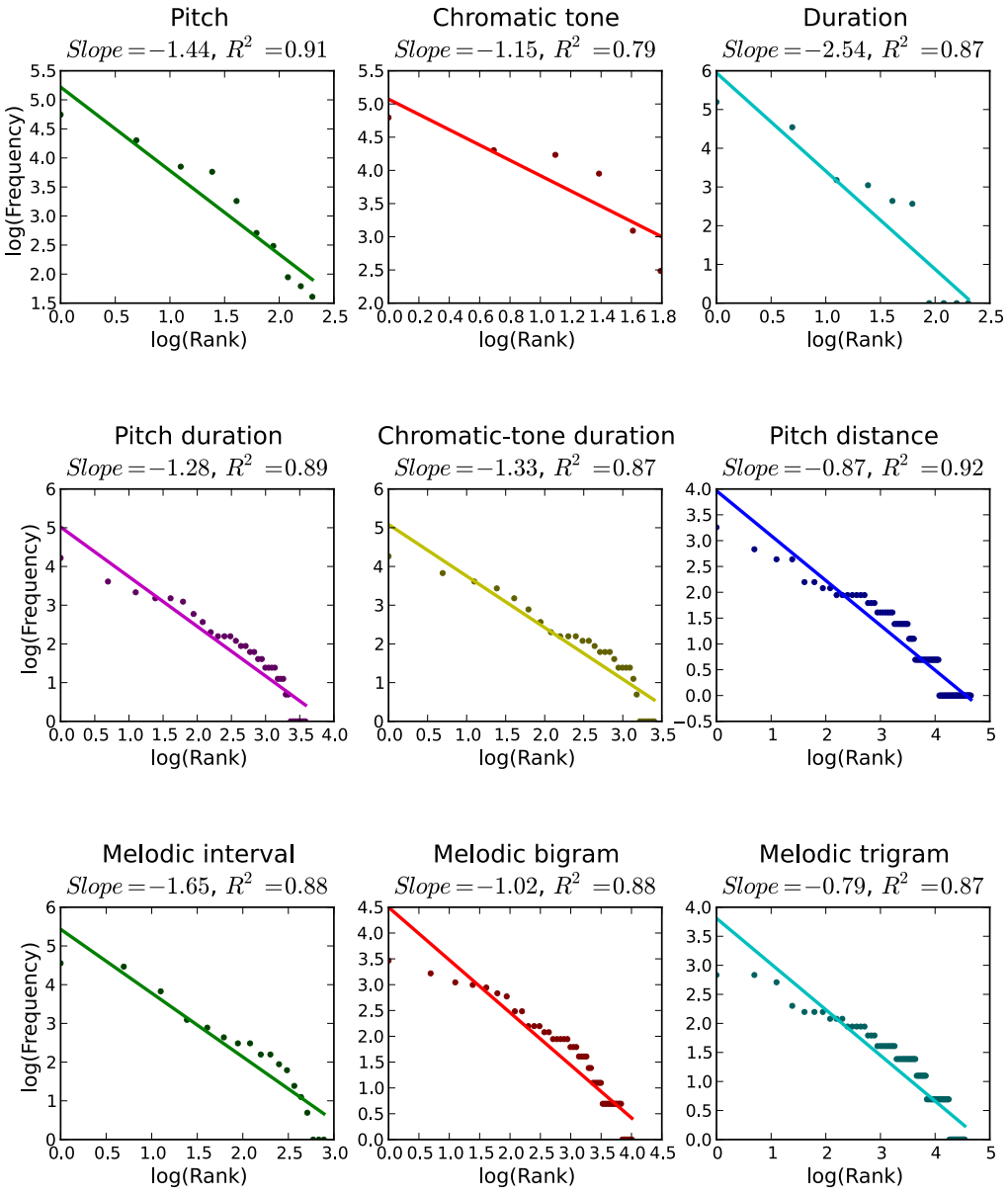


Figure 3.4 Rank-frequency distributions and slopes for each metric applied to *The Beatles' Let It Be*. Most of the metrics display slopes near -1, as predicted by Zipf's Law.

A large corpus of MIDI-encoded music in different styles was analysed by Manaris with the Zipf-based metrics. The results showed that all music pieces displayed many near Zipfian distributions, with strong correlations between the distribution and the linear fit. Non-music (random) pieces exhibited very few (if any) distributions.

These results suggest that Zipf-based metrics capture essential aspects of the scaling properties in music. They indicate that music tends to follow a distribution balanced between chaos and monotony, i.e. between a near-zero slope and a steep slope approaching negative infinity.

Studies showed that different styles of music exhibited different slopes and demonstrated further that the slopes could be used successfully in several music classification tasks. A connection between Zipf metrics and human aesthetics was also revealed (Manaris et al., 2005).

3.3.3 Fitness Function

Since Zipf distributions are so prevalent in existing music, it seems reasonable to assume that new music must also exhibit such properties. The obvious question is then, can a fitness function based on Zipf's Law guide evolution towards pleasant music?

Some research exists in this area: Manaris et al. (2007) performed several music generation experiments using Zipf metrics for fitness. Different melodic genes were used for the initial population and successful results were reported when an existing music piece was used. In other words, *variations* of existing music were evolved. The work presented herein, however, is focused on creating new music from scratch.

Each Zipf-based metric extracts a *slope* from a music piece. Assume that the value of some favourable slope is known a priori – that is, a *target* slope which evolution should search for. The fitness is then a function of the distance (error) to the target.

For a single metric, the *target fitness* is defined as a Gaussian:

$$f_m(x;T) = e^{-(\frac{T-x}{\lambda})^2} \tag{3.2}$$

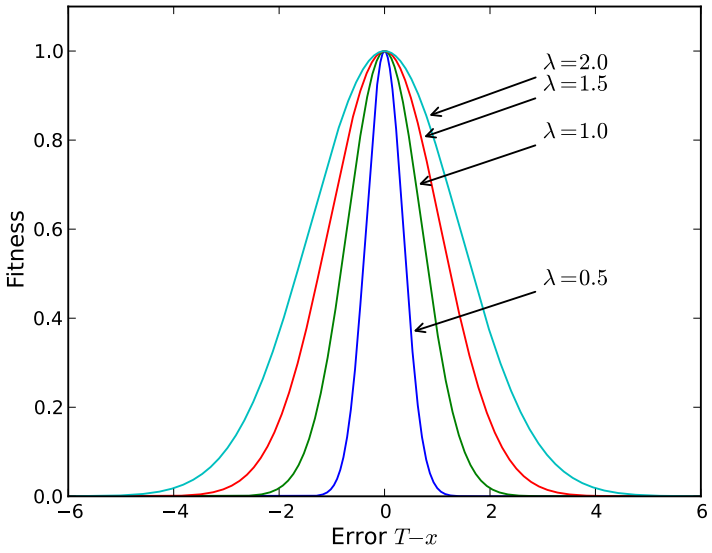


Figure 3.5 Fitness plot of the target fitness f_m for different tolerance values λ.

where m denotes the metric, T is the target slope for the given metric, x is the metric slope of some evolved music piece and λ is the tolerance – a positive constant. f_m results in smooth fitness values ranging from 0.0 (when $|T - x|$ is above some threshold) and 1.0 (when $T = x$). The tolerance λ adjusts this threshold (and the steepness of the fitness curve). Fitness will approach zero when $|T - x|$ is approximately 2λ. Figure 3.5 shows the fitness curves of f_m for different tolerance values.

Of course, a single metric is unlikely to be sufficient alone as the fitness function. Several metrics should be taken into account. Thus instead of a single target slope, a *vector* of target slopes is used. Combining the target fitness of several different metrics as a weighted sum gives:

$$f(\mathbf{x}; \mathbf{T}) = \sum_{i=1}^{N} w_i f_i(x_i; T_i) \tag{3.3}$$

where N is the number of metrics, i denotes the metric number, w_i its weight and f_i the single metric target fitness function in equation (3.2). Finally f is normalized to produce fitness values in the interval $[0, 1]$.

3.4 Fitness Based on Distribution Similarity

Experiment results from Chapter 4 demonstrate that Zipf metrics can be used successfully as fitness for evolution of pleasant melodies. Some musical knowledge was necessary for evolution to produce coherent results, e.g. constraints in the form of a scale, the number of possible pitches and so on. However, a majority of the evolved melodies were in fact rather unpleasant. As discussed in Section 4.4, Zipf metrics capture scaling properties only, which were shown to be insufficient for pleasant music alone.

Zipf's Law in music seems to be *universal* in that it applies to many different (if not all) styles of music. Musical taste, however, varies greatly from person to person and depends on many factors such as nationality, culture and musical background. Exposure to music will likely affect our musical taste, e.g. an Indian is likely to prefer Indian music over country, simply because he is more familiar with the style.

Thus, instead of attempting to model universal musical properties, it might be more fruitful to model properties in *certain styles* of music.

Musicians usually focus on a few selected musical styles, but how does creativity come to the musician? An important part of the music creative process is undoubtedly listening to a lot of music for *inspiration*. Musical concepts and ideas are borrowed from music we like, either knowingly or subconsciously. Either way, there is certainly an element of *learning* involved in musical creativity.

This chapter presents a more knowledge-rich approach to musical fitness. The approach takes both *contents* and scaling properties into account, which can be *learned* from existing music pieces.

3.4.1 Metric Frequency Distributions

Each metric (see Section 3.2.1) counts the occurrence of some type of musical event, producing a *frequency distribution* of events. Figure 3.6 shows the frequency distribution of chromatic tones used in *Mozart's Piano Sonata No. 16 in C major (K. 545)*. From the distribution it is seen that the piece mainly concerns the pitches C (0), D (2), E (4), ..., i.e. the C major scale.

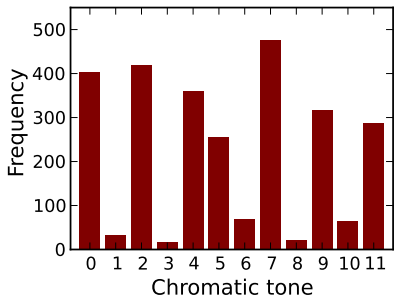


Figure 3.6 Frequency of chromatic tones from *Mozart's Piano Sonata No. 16.*

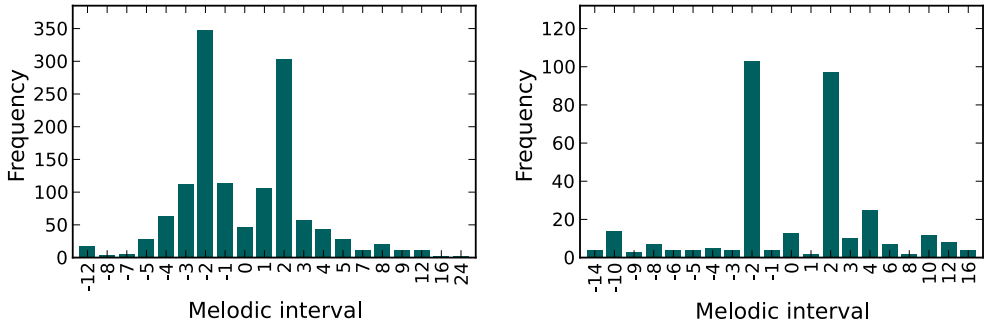


Figure 3.7 Melodic intervals from *Mozart's Piano Sonata No. 16* (a) and *Debussy's Prélude Voiles* (b). Note the difference in shape and intervals used.

Perhaps more interesting is the frequency of melodic intervals (Figure 3.7a): the major (±2) and minor second (±1) intervals dominate the melody, followed by the minor third (±3), unison (0), perfect fourth (±5) and major third (±4). Compare this to the melodic intervals in *Debussy's Prélude Voiles (Book I, No. 2)* shown in Figure 3.7b, where the major second is mainly used – evidence of the whole tone scale that is employed.

As can be seen, there is a wealth of knowledge in such frequency distributions. A fitness function which makes use of this knowledge could steer evolution towards music that is statistically *similar* to some selected piece.

3.4.2 Cosine Similarity

The fitness function presented herein takes as input a set of discrete target frequency distributions, which can be learned from existing music. The

fitness score is calculated based on the similarity to these distributions.

Concepts from the field of Information Retrieval (IR) are borrowed, where a common task is to score documents based on similarity. The standard technique operates on the frequency of different words in a document – *term frequencies*. Each document is viewed as a *vector* with elements corresponding to the frequency of the different terms in the dictionary. With the document vector model, the similarity of two documents can be assessed by considering the *angle* between their respective vectors – the *cosine similarity*:

$$sim(\mathbf{A}, \mathbf{B}) = cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|} \tag{3.4}$$

A and **B** are the two document vectors, the numerator is their dot product and the denominator is the multiple of their norms. Since the vector elements are strictly positive, the similarity score ranges from 0 meaning completely dissimilar (independent) to 1 meaning exactly the same. The cosine similarity has the advantage of being unaffected by differences in document length – the denominator normalizes the term frequencies.

In the musical domain, the documents are music scores. Instead of words, musical features are considered: pitches, melodic intervals, rhythm etc. as described in Section 3.2.1. For instance, the cosine similarity between the melodic intervals in Mozart's and Debussy's pieces (Figure 3.7) is 0.92.

3.4.3 Fitness Function

For a given metric m, fitness is defined as the cosine similarity between the frequency vectors of the music individual $\mathbf{x}$ and a *target piece* $\mathbf{T}$:

$$f_m(\mathbf{x}; \mathbf{T}) = sim(\mathbf{x}, \mathbf{T}) = \frac{\mathbf{x} \cdot \mathbf{T}}{\|\mathbf{x}\| \, \|\mathbf{T}\|} \tag{3.5}$$

f_m will thus reward music with features that are statistically similar to the target piece with respect to the metric. In other words, music which exhibits the same events at similar relative frequency. The target vector can stem from a single music piece or a collection of pieces.

For example, if the melodic intervals in Figure 3.7b were used as the target vector $\mathbf{T}$, the fitness function would reward music in the whole tone scale similar to *Debussy's Prélude Voiles*. Furthermore, positive intervals are approximately as frequent as negative intervals. Balanced melodies would therefore be favoured, i.e. where upward and downward motions occur approximately as often.

For multiple features, the fitness is simply the weighted sum of the similarity scores f_m:

$$f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N; \mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_N) = \sum_{i=1}^{N} w_i f_i(\mathbf{x}_i; \mathbf{T}_i) \tag{3.6}$$

where i denotes the metric, $\mathbf{x}_i$ and $\mathbf{T}_i$ are the metric frequency vectors and w_i is the weight (importance) of metric i. For convenience, the sum is normalized to produce fitness values in the range $[0, 1]$.

As mentioned, the fitness function rewards music which exhibits properties that are statistically similar to a target music piece. The motivation for this approach is not to copy, but rather to *learn* from existing music by extracting common music knowledge plus a bit of *inspiration*. The amount of inspiration depends on which metrics are included. For instance, higher level melodic n-grams will reward music which mimics the melody in the target piece.

3.4.4 Relationship to Zipf's Law

Zipf's Law applies to rank-frequency distributions, i.e. only the relative frequencies of events are considered. Cosine similarity, on the other hand, operates directly on frequency distributions and thus both relative frequency and event content is taken into account. That is, cosine similarity incorporates Zipf's Law.

If the cosine similarity of two frequency distributions A and B is 1.0, it follows that their respective rank-frequency distributions will have the same shape. Consequently their Zipf slopes will also be identical:

$$sim(A, B) = 1.0 \Rightarrow slope(A) = slope(B)$$

Assuming that the target music piece exhibits Zipfian slopes, the similarity-based fitness function (3.6) will thus promote music with similar slopes.

3.4.5 Filtering

When counting the many events in real-world music, there is likely to be some events that occur very rarely, i.e. have low frequencies. In other words, there is bound to be some *noise*. When target vectors are derived from a music score, it is desirable to put emphasis on the most descriptive events. Given the repetitive nature of music, it is reasonable to assume that important events occur often. That is, events with very low frequencies are considered of little value and can be filtered out.

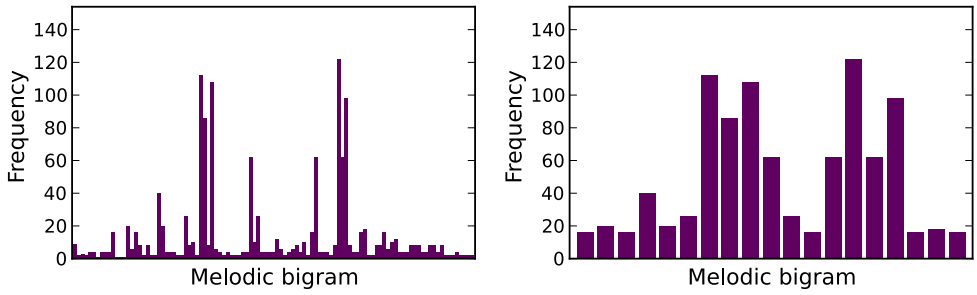
A simple filtering method is to discard events whose frequency is below some *threshold*. This has proven to be an effective method in text categorization, where the dimensionality of document vectors can be greatly reduced by isolating the most descriptive terms (Yang and Pedersen, 1997).

When real-world music is used for target vectors, events are filtered out whose normalized frequency is below a threshold according to the criterion:

$$\frac{f}{N} < p \tag{3.7}$$

where f is the event frequency, N is the total number of events in the score and p is the threshold in percent. For example, a threshold of $p = 0.01$ would discard events whose frequency accounts for less than 1% of all events.

Figure 3.8a shows the frequency distribution of melodic bigrams from *Mozart's Piano Sonata No. 16*. Applying a 1% threshold filter results in the removal of 86 events, producing the much smaller distribution with 19 events as seen in Figure 3.8b.



(a) **(b)**

Figure 3.8 Frequency of melodic bigrams from *Mozart's Piano Sonata No. 16*: full distribution (a) and after filtering events below a 1% threshold (b).

Chapter 4

Experiments: Zipf's Law

Previous work by the author explored the use of Zipf's Law as fitness for evolution of short melodies (Jensen, 2010). Results showed that pleasant melodies could indeed be generated with such a technique. Several favourable musical features were seen in the results including melodic motifs. Some constraints were necessary to achieve any pleasant results, most notably restricting note pitches to a pre-defined scale.

Several different target slopes were tried in order to improve the quality of the evolved melodies, but it was difficult to find a good set of slopes. On average, only 10% of the evolved melodies were perceived as pleasant. Furthermore, the melodies lacked several important musical features, including rhythm and structure.

In this chapter, experiments are presented where the goal was to improve these results. In Section 4.1, the performance of the *tree-based* representation is investigated. Melodies evolved with the tree-based representation are qualitatively compared to melodies from previous work in Section 4.2. Finally, rhythmic qualities are introduced in Section 4.3.

4.1 A Musical Representation

In earlier work (Jensen, 2010), three *linear* genotypes were explored with respect to the Zipf-based fitness function: An event-based binary genotype,

an event-based vector and a dynamic vector representation. All of these representations fall under the GA umbrella and the event-based vector achieved the best fitness of the three. However, a near-maximum fitness was never achieved and convergence was relatively slow.

Experiments were therefore performed in an attempt to *improve fitness* and *convergence speed*. Two options were investigated:

1. Test a new *tree-based* genotype.
2. Modify the old vector-based genotypes.

4.1.1 Introduction

As discussed in Section 2.5, two approaches to representation are commonly found in the evolutionary music literature. The first is the linear binary/vector genotype (GA) similar to what was already tried. The second is the tree-based representation (GP), which some have argued is well suited for music because of its hierarchical structure (see Section 3.1). It was therefore decided to test a GP approach to see if it improved fitness and convergence speed.

In summary, the event-based genotypes from previous work are similar to the example shown in Figure 2.4, i.e. employing implicit note durations. However, the event-based vector is symbolic instead of binary and consistently achieved better performance. The dynamic vector genotype is simply a list of (enable, pitch, duration) triplets, i.e. with explicit note durations. The first element, enable, is a flag which dictates whether the triplet should be evaluated or ignored when interpreted by the phenotype. This allows for variation in the number of notes.

The hypothesis was that the slow evolutionary convergence of the vector-based genotypes was caused by low genetic diversity in the population. As discussed in Section 2.2, diversity is a key element in Evolutionary Computation. The two vector-based genotypes were changed to see if this was the case. As a first test, the mutation operator was modified so as to change an *entire gene* instead of a single gene element. Another mutation variant was also tried, where mutation changed a whole genome *segment*. Finally, the *mutation rate* was increased to see if it improved the performance.

4.1.2 Setup

For all the experiments, evolution was run for 500 generations. For each generation, the maximum fitness was averaged over 30 runs and plotted. The choice of parameters is described in more detail in Section 4.1.3.

General Parameters

The evolutionary parameters listed below were identical for all experiments unless otherwise noted:

- Population size: 100
- Mutation rate: 0.1
- Crossover rate: 0.9
- Tournament selection: $k = 5$, $e = 0.1$ (see Section 2.2)
- Fitness: Weighted sum with 10 Zipf metrics:
  - Metrics: pitch, chromatic-tone, duration, pitch duration, chromatic-tone duration, pitch distance, chromatic-tone distance, melodic interval (absolute), melodic bigram (absolute), melodic trigram (absolute)
  - Target slopes: $T_i = -1.0$
  - Tolerance: $\lambda = 0.5$
  - Weights: uniform 1.0 for all metrics

Tree-Based Representation Parameters

The parameters for the tree-based representation (see Section 3.1) were selected to closely match the properties of the vector-based genotypes:

- Pitches: 12
- Durations: 5

- Resolution: 16
- Functions: concatenation (+)
- Terminals: *simple* (pitch, duration) or *event-based* (event-type, value)
- Maximum tree depth: 5 or 6 resulting in maximum $2^5 = 32$ or $2^6 = 64$ notes, respectively
- Initialization method: *grow* or *full*
- Function probability: 0.9 (default)
- Terminal probability: 0.1 (default)

The event-based terminal scheme is similar to the event-based vector, i.e. tuples of (event-type, value) where event-type is either *note* or *hold* and the value holds the pitch.

Vector-Based Representation Parameters

The parameters for the vector-based representations were derived from the previous work and are summarized below.

For the *event-based vector*, the parameters were:

- Bars (measures): 4
- Pitches: 12
- Resolution: 16

The following parameters were used for the *dynamic vector*:

- Length: 64
- Pitches: 12
- Durations: 5
- Resolution: 16

4.1.3 Experiment Setup

In total, 14 experiment runs were performed where different parameters were tested. In six of the runs the tree-based genotype was used, while in eight runs the vector-based genotypes were employed. The setup for each experiment is detailed below.

Tree-Based Representation

For the tree-based representation, it was important that the parameters were as similar to the vector-based genotype as possible. This was both to ensure a fair comparison, but more importantly to isolate whether the tree *structure* was beneficial for music.

As such, two different *terminal schemes* were tested: simple and event-based, similar to the dynamic vector and event vector, respectively.

Another key parameter is the *maximum tree depth*, which dictates the maximum number of possible notes. Fewer notes results in less freedom for evolution to find melodies with high fitness. It was therefore important that the max-depth parameter was roughly equivalent to the number of possible notes in the vector-based genotypes: 64. The max-depth was thus set to 6, resulting in $2^6 = 64$ number of notes. It was decided to also try a depth of 5 (32 notes) to see how the tree-based genotype would perform with less freedom than the vectors.

Finally, two different tree *initialization methods* were tested: grow and full, to see if either method led to significant advantages with respect to fitness and convergence speed.

As such, for both simple and event-based terminals, the following configurations were tested:

1.1. Max tree depth: 6, initialization method: *grow*

1.2. Max tree depth: 6, initialization method: *full*

1.3. Max tree depth: 5, initialization method: *grow*

Resulting in a total of 6 experiment runs.

Vector-Based Representation

As mentioned, it was hypothesized that the genetic diversity was too low, causing slow convergence for the vector-based genotypes. The mutation operator previously employed was designed to mutate only part of the gene, i.e. a single element of the tuple or triplet.

To increase genetic diversity, the mutation behaviour was changed so that the entire gene was randomly changed, i.e. *all* elements of a tuple (triplet). A second mutation variant was also explored where a whole genome segment is altered, i.e. *multiple* sequential tuples (triplets), in order to boost genetic diversity even more.

As a final measure, the *mutation rate* was increased to see if it improved the performance. However, it was decided to only test the increased mutation rate using the mutation operator with the best performance.

Thus, for both the event-based vector and dynamic vector, the following tests were performed:

2.1. Mutation of entire gene

2.2. Mutation of genome segment

2.3. Using the best mutation operator from (2.1) and (2.2), increase mutation rate to:

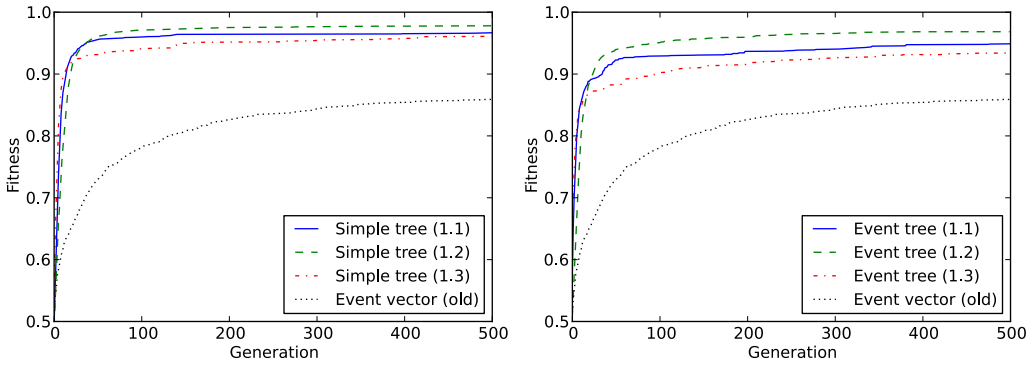
   (a) 0.5

   (b) 0.9

This resulted in a total of 8 experiment runs.

4.1.4 Results and Discussion

The results from the 14 experiments are presented below and compared to earlier results. Finally, the best configurations of the three genotypes are presented.



(a) Simple terminals **(b)** Event-based terminals

Figure 4.1 Fitness plots for the three different configurations of tree-based genotypes with simple (a) and event-based (b) terminals. The event vector from earlier work is also shown as reference (old).

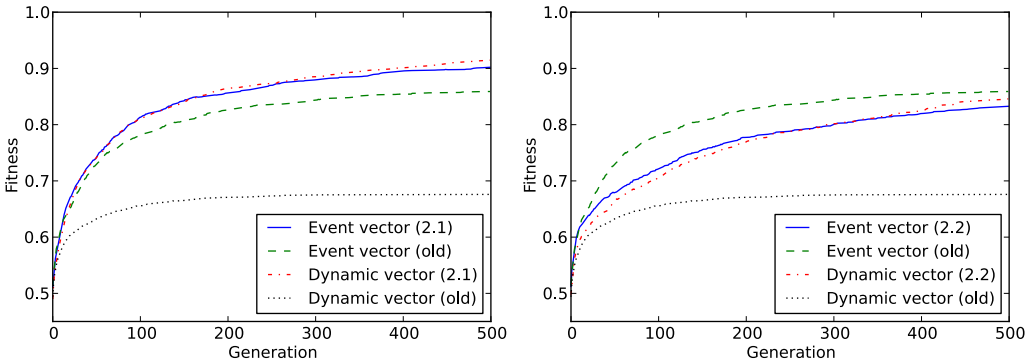
Tree-Based Representation

Figure 4.1 shows the fitness plots from the 6 different tree-based configurations tried, along with the old event vector from earlier work as reference. Surprisingly, all variants of the tree-based genotypes yielded superior performance compared to the vector representation. On average they converged much faster and reached a higher maximum fitness.

From the fitness plots, it is evident that the best performer was the tree with *simple* terminals (Figure 4.1a), which consistently reached better results than the *event-based* counterpart (Figure 4.1b). Trees with a maximum depth of 5 also performed well (configuration 1.3), which signifies that even short melodies could achieve a high fitness.

As can be seen, using the *full* method for initialization (1.2) seemed to result in marginally higher fitness values. However, the melodies evolved using the *full* method were generally longer than those evolved with *grow* (1.1): the average number of notes was 45 and 29 for *full* and *grow* respectively (with *simple* terminals). As such, the slight difference in fitness values was likely related to melody length rather than the initialization method itself.

A similar trend was found with the *event-based* tree, where the average number of notes was 20 (*full*) and 16 (*grow*). This indicates that the worse performance found with the event-based terminals was also due to the shorter



(a) 2.1. Mutation of entire gene **(b)** 2.2. Mutation of genome segment

Figure 4.2 Fitness plots for the vector-based genotypes using two different mutation operators: 2.1. Mutation of entire gene (a) and 2.2. Mutation of genome segment (b). For reference, the plots using the old mutation operator also shown.

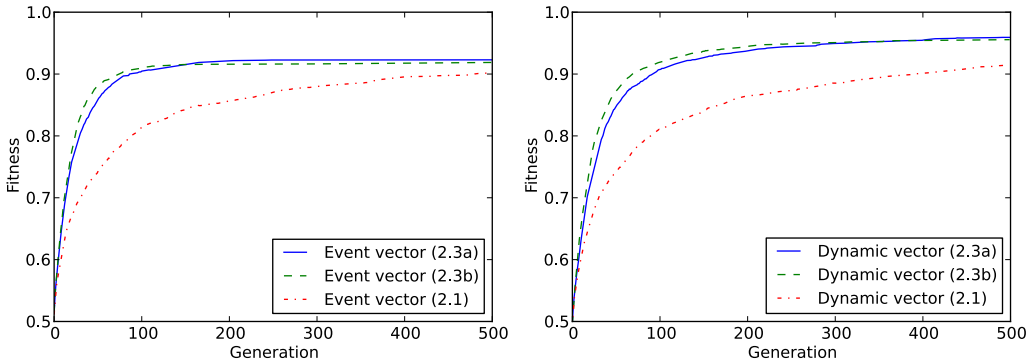
melody lengths. Thus there seemed to be no significant difference between the two terminal schemes, at least from a fitness perspective.

At this point, it was not evident why the tree-based representations performed so much better than the vectors. Apart from the difference in structure, the mutation operator used for GP is more explorative than the GA's, in that it can alter many genes (nodes) at the same time as opposed to only one. It was thus hypothesized that mutation was a key element to the success of GP.

Vector-Based Representation

Figure 4.2 shows the fitness plots from the two mutation operators applied to both vector-based genotypes: 2.1. Mutation of entire gene (4.2a) and 2.2. Mutation of genome segment (4.2b). The fitness from using the old mutation operator are also included for reference.

As shown in Figure 4.2a, mutation of the entire gene (2.1) resulted in a slight improvement in fitness when applied to the event vector. For the dynamic vector, however, entire-gene mutation drastically improved the performance; much faster convergence and higher fitness was achieved, matching the performance of the event-vector. These results confirmed the hypothesis that more genetic diversity would improve performance. However, the fitness was still not comparable to the tree-based representations.



(a) Event vector **(b)** Dynamic vector

Figure 4.3 Fitness for the two vector-based genotypes using entire-gene mutation (2.1) and increased mutation rates 0.5 (2.3a) and 0.9 (2.3b).

The fitness plots from genome segment mutation (2.2) can be seen in Figure 4.2b. For the event vector, segment mutation lead to *worse* performance compared to the old mutation operator, i.e. mutation was destructive in this case. The dynamic vector displayed a performance increase, but results were not as good as 2.1.

Since entire-gene mutation (2.1) resulted in better performance than segment mutation (2.2), it was therefore used in test 2.3: increased mutation rate of 0.5 (2.3a) and 0.9 (2.3b). In Figure 4.3, the fitness plots from the four runs may be seen, along with the results from experiment 2.1 for comparison.

As can be seen, a mutation rate of 0.5 further improved the performance of both vector representations, with the dynamic vector now surpassing the event vector. Increasing the mutation rate further to 0.9 did not seem to improve results at this point.

A Final Comparison

Figure 4.4 shows the fitness plots for each genotype configuration yielding the best performance, where the tree-based representation is the clear winner. The genotype configurations are summarized below:

Tree: *simple* terminals, max-depth: 6, initialization method: *full* (1.2)

Vectors: *entire-gene* mutation, mutation rate: 0.5 (2.3a)

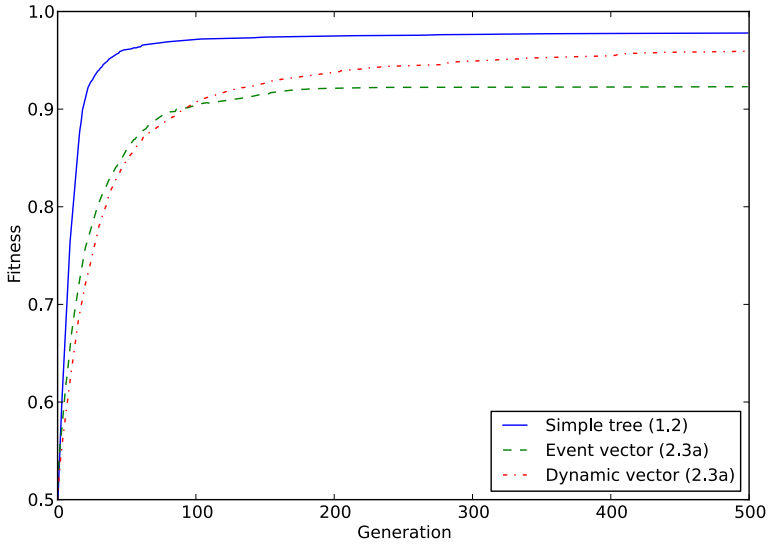


Figure 4.4 The best configurations found for the tree-based and vector-based representations.

At this point, no further improvements were made. Why the GP approach performed so well is not fully understood. One hypothesis is that the modular, hierarchical structure of the tree is particularly beneficial for music, at least with respect to Zipfian properties. Another reason might be that trees allow genomes of different lengths, while the vectors explored here did not.

Either way, the primary focus of this research is *fitness* for music and it was therefore decided to use the tree-based genotype for future experiments.

It should be noted that these experiments by no means represent an exhaustive comparison of GA versus GP for music. It is almost certain that a GA expert would be able to design a genotype which is better able to match performance of the GP presented herein. As such, these experiments are primarily included to justify the choice of a tree-based representation for the rest of this research.

4.1.5 Summary

The tree-based representation was shown to outperform the linear representations previously employed. Although improvements were achieved with the vector-based genotypes, the tree was still clearly the best performer with

higher fitness and faster convergence speed. Why the tree-based representation performed so well is not fully understood. It is hypothesized that the modular, hierarchical structure of the tree is particularly beneficial for music.

4.2 Tree-Based Composition

In the previous experiment, tree and vector-based representations were compared with respect to fitness performance. The results revealed that the tree was able to reach a higher fitness than both vector-based representations. However, such results do not necessarily imply that the tree-based genotype yields more pleasant melodies.

4.2.1 Introduction

To test the ability of the Zipf-based fitness function to model music pleasantness, 30 melodies were evolved using the tree-based representation. A qualitative analysis of the melodies was performed and compared to the old vector-based results from Jensen (2010). The old melodies exhibited lower fitness values than the results from Experiment 4.1. Some of the problems found in these melodies were long repetitions of the same note and a bias towards short notes.

4.2.2 Setup

As before, the parameters were chosen to be similar to what was used in the previous work (Jensen, 2010):

- Pitches: 8 (one octave)
- Durations: 4
- Resolution: 16
- Functions: concatenation (+)
- Terminals: *simple* (pitch, duration)

- Maximum tree depth: 6
- Initialization method: *grow*
- Scale: C major scale

Otherwise, the parameters were the same as in Section 4.1, i.e. 30 runs of 500 generations each. The most fit melody was picked out at the end of each run.

4.2.3 Results and Discussion

Table 4.1 lists some interesting statistics from the 30 melodies evolved using the tree-based representation and the (old) event vector. As can be seen, the melodies evolved with the tree have significantly higher fitness than that of the event vectors, with both a higher mean and lower standard deviation. The two representations resulted in approximately the same number of notes. However, the tree-based melodies exhibit a much larger variation ($\sigma = 16.6$).

A qualitative comparison of the tree-based and event vector melodies revealed that the former indeed sounded more musical. For instance, they exhibited more motifs and repetitions, which made the music sound more coherent. In addition, the melodies contained less of the overly repetitive sequences found in the event-vector melodies, even though they were evolved using the same target slopes of -1.0. Finally, since the tree-based representation make use of explicit note durations (not event-based), there was no bias towards short notes. This was seen as another favourable property.

Figure 4.5 shows one of the most pleasant melodies evolved – dubbed "Zelda". Notice the two recurring motifs marked in squares and circles.

Table 4.1 Evolved melody statistics using the old event vector and tree-based representation: mean (standard deviation, σ).

	Event vector	Tree
Fitness	0.87 (0.05)	0.97 (0.01)
Number of notes	33.7 (3.7)	36.2 (16.6)
Melody length	16 (0) (fixed)	32 (13.3)

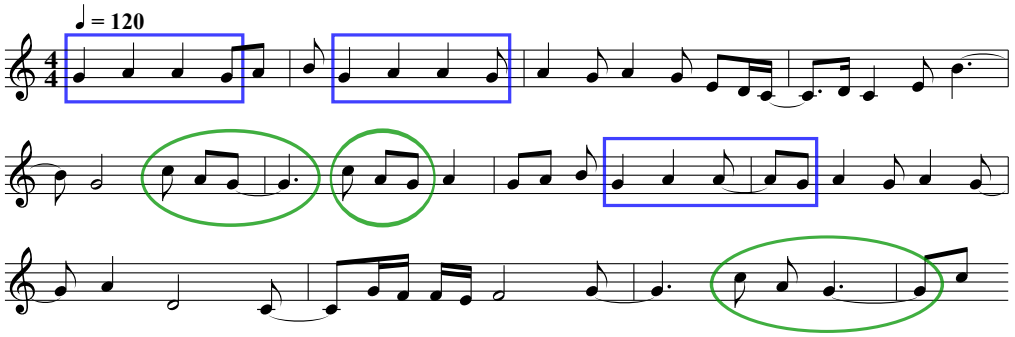


Figure 4.5 Pleasant evolved melody with the tree-based genotype – "Zelda".

However, although pleasant music was evolved, many of the melodies were still rather unpleasant. A common feature was the lack of consistent *rhythm*, which resulted in a rather choppy musical flow.

4.2.4 Summary

The tree-based representation indeed resulted in more pleasant melodies than previously achieved with a vector-based genotype. An increased number of melodic motifs and pleasant repetitive features improved their musical qualities. However, lack of consistent rhythm was a common cause of unpleasantness.

4.3 Adding Rhythm

As stated, a characteristic of the evolved melodies from Experiment 4.2 was the lack of a consistent rhythm. Of the ten metrics used for fitness so far, seven measure some property related to note *pitches*. Only three operate on note *durations*, which is the main ingredient in rhythm. Furthermore, none of these duration metrics take contextual information into account. It is thus perhaps not so surprising that the evolved melodies lacked rhythmic features.

In this experiment, four new metrics related to rhythm are explored.

4.3.1 Introduction

Rhythm, *rhythmic interval*, *rhythmic bigram* and *rhythmic trigram* were the four new metrics that were tested (see Section 3.2.1). The hypothesis was that these metrics would endorse consistent rhythmic features, in the same way that metrics based on melodic intervals promoted melodic motifs.

As a first step, a collection of real-world music pieces were analysed with respect to the four rhythm metrics to see if Zipfian distributions were prevalent. Then the melodies evolved in the previous experiment (4.2) were analysed in the same way. This would reveal whether the rhythm metrics captured properties which were not already taken into account by the other metrics.

After the preliminary analysis, a set of melodies were evolved with the rhythm metrics included in the fitness function. The results were then compared to the melodies from Experiment 4.2, to see if there were any improvements.

4.3.2 Experiment Setup

As mentioned, the experiment was twofold. First, a preliminary analysis of real-world music versus evolved music was performed, with respect to the new rhythm metrics. Second, the rhythm metrics were included in the fitness function and a new set of melodies were evolved.

1. Metric Analysis

For the first step, *Mozart's Piano Sonatas* (19 pieces) were chosen. They can be characterized as having a steady rhythm and easily separable melody. For the melody of each piece, the mean slopes were calculated for each of the rhythm metrics. The same was done for the evolved melodies from Experiment 4.2.

However, Zipf metrics that yield sensible results on a global level might not apply at a *local level* (subsection) of the same music (Jensen, 2010). Therefore, Mozart's pieces were also divided into smaller subsections of approximately the same length as the evolved melodies (8 bars) and were analysed individually for rhythmic slopes.

Table 4.2 Rhythm metric slopes – mean (std. dev.) – from Mozart's Piano Sonatas and Evolved melodies from Experiment 4.2.

	Mozart PS	Mozart PS (local)	Evolved (4.2)
Rhythm	-2.42 (0.14)	-1.85 (0.63)	-1.02 (0.06)
Rhythmic interval	-1.52 (0.13)	-1.46 (0.52)	-1.05 (0.41)
Rhythmic bigram	-1.19 (0.08)	-1.02 (0.41)	-0.81 (0.18)
Rhythmic trigram	-1.01 (0.07)	-0.80 (0.38)	-0.63 (0.23)

2. Melody Generation

A set of 30 melodies were evolved using the same setup as the previous experiment, but with the four rhythm metrics included. In addition, the duration metric was *removed*, since it is identical to *rhythm* when there are no rests (which is the case for the representation used). The target slopes T_i remained the same: uniform -1.0 for all metrics. In summary, fitness was composed of 13 Zipf metrics:

- Metrics: pitch, chromatic-tone, pitch duration, chromatic-tone duration, pitch distance, chromatic-tone distance, melodic interval (absolute), melodic bigram (absolute), melodic trigram (absolute), *rhythm*, *rhythmic interval*, *rhythmic bigram*, *rhythmic trigram*
- Target slopes: $T_i = -1.0$
- Weights: uniform 1.0 for all metrics

Otherwise, the setup was the same as in Experiment 4.2.

4.3.3 Results and Discussion

The results of the experiment are presented in the following two sections: an analysis of the rhythm metrics and the new set of evolved melodies.

1. Metric Analysis

Table 4.2 lists the mean slopes (and standard deviation) from *Mozart's Piano Sonatas* (global and local) and from the melodies evolved in Experiment

Table 4.3 Slopes and fitness from evolved melodies with rhythm-based metrics.

	Slope mean (std. dev.)
Rhythm	-1.01 (0.09)
Rhythmic interval	-1.05 (0.08)
Rhythmic bigram	-1.02 (0.06)
Rhythmic trigram	-0.86 (0.07)
Fitness	0.96 (0.02)

Section 4.2. As can be seen, the rhythm metrics display near-Zipfian distributions in Mozart's sonatas. Furthermore, the slopes from Mozart's pieces are steeper than in the evolved melodies, both at a global and local level. Recall from Section 3.3.2 that steeper slopes indicate more monotonous (less random) distributions.

In the previously evolved melodies, the *rhythm* slopes were all near -1.0, since this specific metric is identical to *duration* when there are no rests in the music. The other rhythmic metric slopes, however, are flatter when compared to Mozart's pieces, which indicates more randomness in the evolved music. In other words, the evolved melodies lacked important rhythmic features, at least when compared to Mozart's pieces.

2. Melody Generation

In Table 4.3, slope and fitness statistics from the 30 melodies evolved with the rhythm-based metrics are presented. As can be seen, evolution was able to fully optimize all the rhythm metrics, except the rhythmic trigram. Compared to the rhythmic slopes of the evolved melodies in Table 4.2, the new melodies exhibited generally steeper slopes. In other words, less random rhythmic features. Fitness was also consistently high across all the runs.

When listening to the evolved melodies, it was evident that they indeed exhibited a more consistent rhythm than earlier results. An example melody is shown in Figure 4.6, where a recurring rhythmic motif is found (♩♪♪♩). Unfortunately, a majority of the melodies were still rather unpleasant mostly because of the strange melodic intervals used.

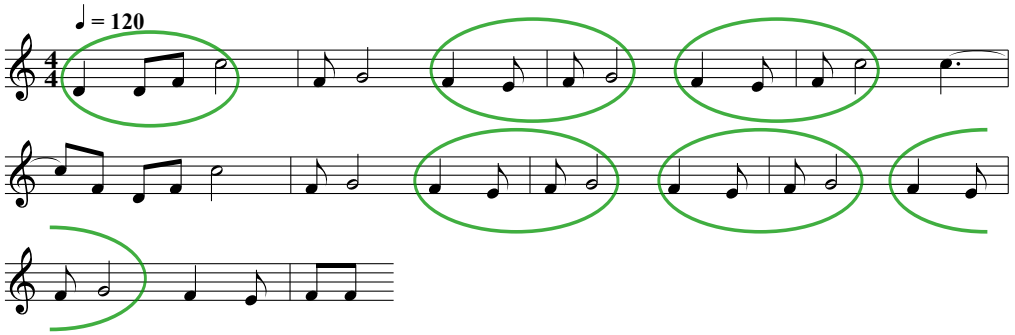


Figure 4.6 Evolved melody with a rhythmic motif (circled).

4.3.4 Summary

The addition of metrics related to rhythmic intervals, bigrams and trigrams did indeed result in a more consistent rhythm in the evolved melodies. Recurring rhythmic motifs were also found, which increased pleasantness.

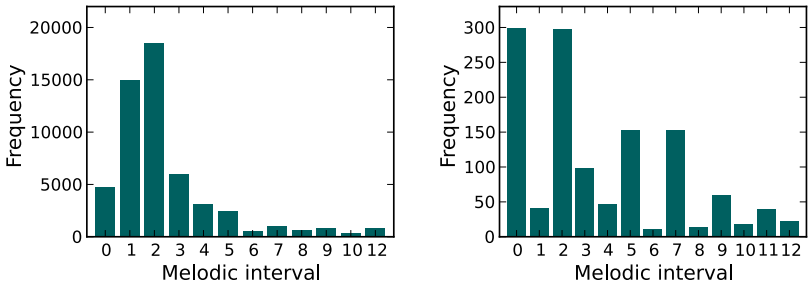
4.4 Conclusions

The experiments and results presented in this chapter demonstrate that Zipf metrics can be used to evolve pleasant music. A tree-based representation was used, which was shown to outperform the linear representations previously employed.

Music evolved using the tree-based approach were generally also perceived as much more pleasant. The addition of metrics for rhythm further improved results. Some musical knowledge was necessary for any pleasant results though (e.g. a pre-defined musical *scale*), which were implemented as constraints in the representation.

However, a majority of the evolved melodies were in fact still rather unpleasant (around 90%). Most notably, the "choice" of melodic intervals was often strange and unmusical. Large, dissonant intervals were common and were perceived as the main source of unpleasantness. It was as if the choice of intervals was completely random, which is not far from the truth.

The poorly selected melodic intervals highlight an important limitation of Zipf-based metrics: they only measure the *scaling* properties of the distributions, ignoring *which* data points (musical events) account for the different



(a) Mozart's Piano Sonatas **(b)** Evolved melodies from 4.3

Figure 4.7 Frequency of (absolute) melodic intervals in real-world music (a) and evolved using Zipf metrics (b). Notice the difference in the employed intervals.

frequencies. In other words, the fitness function is oblivious as to which melodic intervals are used, as long as their rank-frequency distribution displays a Zipfian slope. As such, Zipf metrics are rather knowledge-weak and are insufficient for musical fitness alone.

Plot the frequency of melodic intervals as a bar chart and this problem becomes abundantly clear. Figure 4.7 depicts the 13 most frequent (absolute) melodic intervals in *Mozart's Piano Sonatas* (4.7a) and in the evolved melodies from Experiment 4.3 (4.7b). The intervals used by Mozart follows a bell-like distribution, centred around the *small* intervals. For the evolved melodies, however, the distribution is rather noisy. It is, in fact, uniform random when realising that the evolved music was constrained to a single octave in the C major scale. Consider which intervals are possible and estimate their probabilities, and it becomes clear why certain intervals appear more than others.

There were several options at this point. One possibility was to tweak the target slopes and/or add more metrics. But as discussed above, this would only get us so far – more music knowledge was clearly necessary to increase the amount of pleasant results. Furthermore, experience from earlier work proved that finding a set of good slopes was difficult.

More knowledge could potentially be encoded into the genotype as further constraints. However, selecting what knowledge to incorporate is quite challenging (see Section 2.6.2). Furthermore, hard-coded music knowledge doesn't easily scale to different musical styles.

Another possibility that was considered, was to seed the population with real-world music material. In other words, start with an initial population

of melodies that are already somewhat musical. However, this approach is similar to Manaris et al. (2007) where inconclusive results are reported. Nevertheless, this is a technique which deserves further work.

Finally it was decided to keep focus on the fitness function and try a different, but related approach that incorporates more knowledge based on the *similarity* to a learned frequency distribution. The technique is detailed in Section 3.4 and the next chapter presents the related experiments.

Chapter 5

Experiments: Distribution Similarity

Chapter 5 presents the experiments that were performed with the similarity-based fitness function proposed in Section 3.4. In total, three experiments were carried out.

In the first experiment (5.1) the effects of each metric are analysed with respect to a simple music target. Experiment 5.2 demonstrates how the fitness performance was improved. Finally, in Experiment 5.3, two real-world music pieces are used as inspiration (targets) for the evolution of new melodies.

5.1 Basics

The goal of the first set of experiments presented herein was to see how evolution performed with the similarity-based fitness function. An incremental approach was taken, where initially only a few metrics were included. New metrics were then added gradually. This made it possible to examine how each metric affected the evolved music. In order to test and more easily analyse the results, it was decided to apply a very simple music piece as target.

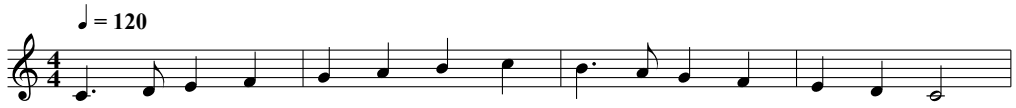


Figure 5.1 Simple melody target, spanning the C major scale.

5.1.1 Introduction

With the incremental approach taken, each sub-experiment was identical except for the number of metrics used for fitness. The initial set of metrics was *pitch* and *chromatic tone*. Then, for each subsequent experiment, a new metric was added to the list. This approach made it easy to compare the evolved melodies from adjacent experiments with respect to the added metric. It also gave an idea of how much knowledge was encoded into each metric distribution.

For each experiment, a hypothesis was formulated in an attempt to predict what musical features each metric would promote. For instance, the pitch metric was thought to constrain the music to a certain scale. The full list of experiments and hypotheses is presented in Section 5.1.3.

The target frequency vectors were extracted from a simple melody spanning the C major scale, simply referred to as the *target* and shown in Figure 5.1. The melody spans the scale up and down, with a few rhythmic features included for good measure. Although rather boring, its simplicity allowed for easier analysis and comparison of the evolved music: *How similar* are the evolved melodies to the target piece? Is the target ever *reproduced* and if so at what point?

5.1.2 Setup

For all the experiments, 30 evolutionary runs were performed, each 500 generations long. At the end of each run, the most fit melody was kept for further analysis.

The evolutionary parameters are listed below:

- Population size: 100
- Mutation rate: 0.1
- Crossover rate: 0.9
- Tournament selection: $k = 5$, $e = 0.1$
- Fitness: Cosine similarity to target piece (see below)
- Representation: Tree (see below)

Fitness

As mentioned in the introduction, the fitness function used for all the experiments was the *weighted sum of metric distribution similarities* – equation (3.6). The metrics explored and general parameters are summarized below.

- Metrics: pitch, chromatic tone, melodic interval, melodic bigram, melodic trigram, rhythm, rhythmic interval, rhythmic bigram, rhythmic trigram
- Target vectors: from simple melody (Figure 5.1)
- Weights: uniform (1.0)
- No filtering

For all the metrics, the target distributions were derived from the simple melody in Figure 5.1. The full set of target distributions can be seen in Figure 5.2 and are frequently referred to in the results.

The alert reader will notice that some of the metrics from Section 3.2.1 have been left out. Regarding the two distance metrics, it was discovered that they produced very flat distributions. In other words, they provided little information and were therefore left out. Pitch duration and chromatic-tone duration were believed to be less important and simply not prioritized.

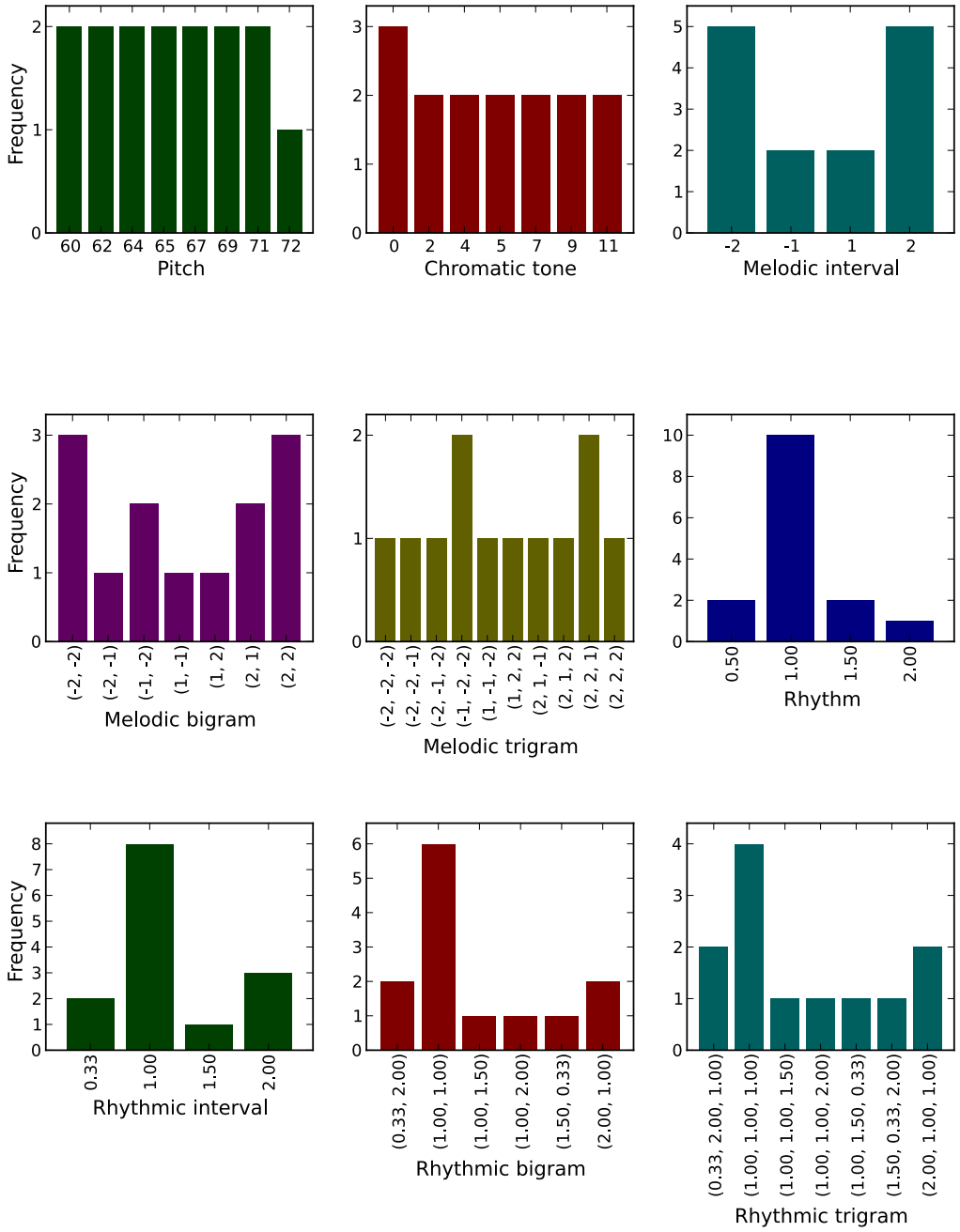


Figure 5.2 Metric frequency distributions from the simple target melody shown in Figure 5.1.

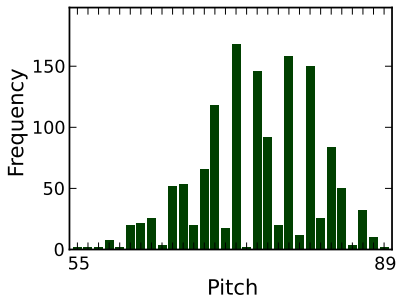


Figure 5.3 Pitch distributions from real-world music tends to have a bell-like shape where the highest and lowest pitches have low frequencies. The distribution shown here is from *Mozart's Piano Sonata No. 16*.

Representation

The genotype and phenotype parameters were chosen to match the length, pitches and durations of the target piece. The target consists of 15 notes, so the maximum tree depth was set to 4, i.e. maximum $2^4 = 16$ notes. The initialization method was set to *full* to generate melodies of similar length as the target piece. The function and terminal probabilities were left at their default values.

The pitches in the target span from 60 (C_4) to 72 (C_5), i.e. one octave. Thus the number of pitches was set to 13 and the pitch reference set to 60. The chromatic scale was used, so all pitches were allowed in the interval $[60, 72]$. This made the fitness function responsible for promoting certain pitches.

Finally, the target contains four different note durations: 1/8 (eight note), 1/4 (quarter note), 3/8 (augmented quarter note) and 1/2 (half note). Since note durations are real-valued, they cannot easily be enumerated (as opposed to pitches). Hence these four durations were used as the duration map.

The restrictions presented above were employed to confine the search space to the bounds of the target piece and thus speed up evolution. The constraints might seem too rigid, however, giving evolution little freedom – e.g. allowing the use of tones outside the pitch range might be beneficial in order to optimize other metrics, such as melodic intervals. However, in the case of real-world music (which is usually much longer than the simple target piece), the distributions tend to have a bell-like shape where the highest and lowest pitches have very low frequencies (Figure 5.3). Therefore these restrictions are in fact significantly less strict when longer target pieces are considered.

The full set of representation parameters are summarized below:

- Functions: concatenation (+)
- Terminals: *simple* (pitch, duration)
- Initialization method: *full*
- Max tree depth: 4
- Pitches: 13
- Pitch reference: 60
- Duration map: $\frac{1}{8}$, $\frac{1}{4}$, $\frac{3}{8}$ and $\frac{1}{2}$
- Scale: chromatic (no scale)
- Function probability: 0.9
- Terminal probability: 0.1

5.1.3 Experiment Setup

A total of eight experiments were performed, where in each experiment a new metric was added to the fitness function. For each metric, a hypothesis was formed regarding what main effect it would have on the evolved music.

The following eight sections cover the setup for the experiments 1 through 8 as well as their respective hypotheses.

1. Pitch and Chromatic Tone

Evolved music with similar pitch or chromatic tone distributions as the target will necessarily employ the same musical scale. A similar pitch distribution also signifies that the tones played are concentrated in the same pitch region.

In order to test whether the evolved music was indeed limited to a similar pitch region, the number of pitches allowed in the genotype was expanded. Thus for this particular experiment, an octave of padding (12 pitches) was added above and below the pitch range, resulting in 37 pitches and a pitch

reference of 48. It was thus expected that the notes would be located exclusively in the centre octave, as in the target piece, and that the top and bottom octaves (the padding) would be unused.

Hypothesis: Confines music to the *scale* and *pitch region* of the target piece.

Although in the correct scale, it was expected that the *order* of the notes would be completely random.

2. Melodic Interval

It was decided to use the relative (non-absolute) version of the melodic interval, since it accommodates more knowledge. Relative intervals convey information about melody direction and balance, e.g it is likely that certain intervals are used more frequently when moving up than down and vice versa.

The two melodic intervals found in the target piece are the major (± 2) and minor second (± 1). Thus it was expected that the inclusion of melodic intervals would result in melodies with similarly small changes in pitch.

Hypothesis: Ensures melodic locality – small changes in pitch.

It was however unknown how similar the evolved melodies would be to the target.

3. Melodic Bigram

A melodic bigram describes two melodic intervals, i.e. three successive notes. There are seven such bigrams in the target piece and each describe a three-note *motif*.

Hypothesis: Promotes similar short melodic motifs (3 notes).

With the inclusion of melodic bigrams, it was expected that the evolved melodies would be more similar to the target piece.

4. Melodic Trigram

Melodic trigrams consist of three melodic intervals – four successive notes. Ten trigrams are found in the target piece which contain four-note motifs.

Hypothesis: Promotes melodic motifs (4 notes) derived from the target.

With the addition of melodic trigrams, the prediction was that the evolved melodies would be very similar to the target piece.

5. Rhythm

Up to this point, note durations were not considered at all in the fitness function. It was therefore expected that the durations were randomly chosen by evolution. Adding rhythm to the fitness function would change this. The target melody is dominated by quarter notes, so evolved melodies with similar rhythm distributions would follow this trend.

Hypothesis: Enforces a similar *tempo*.

Although with a similar tempo, it was anticipated that the melodies would contain a few of the other, less frequent durations scattered around.

6. Rhythmic Interval

The rhythmic interval measures the relationship between the duration of two adjacent notes. It was therefore predicted to see some consistency in the order notes appear. For instance, as in the target piece it was expected to see a $3/8$ note followed by a $1/8$ note, but not the other way around.

Hypothesis: Promotes rhythm consistency.

It was however unknown how similar the evolved rhythm would be to the target.

7. Rhythmic Bigram

Rhythmic bigrams capture the relationship between three adjacent note durations. For example, the first three notes of the target piece (Figure 5.1) forms a rhythmic motif which is repeated twice: ♩.♪♩. With the addition of rhythmic bigrams, it was expected to see motifs like these in the evolved music.

Hypothesis: Promotes short rhythmic motifs (3 notes).

8. Rhythmic Trigram

The relationship between four adjacent note durations conveys even more information about rhythm. The first four notes of the target piece (♩.♪♩♩) form a longer rhythmic motif, which hopefully also would appear in the evolved music.

Hypothesis: Promotes rhythmic motifs (4 notes).

At this point, it was anticipated that the rhythm would be very similar to the target piece.

5.1.4 Results and Discussion

Table 5.1 lists relevant data from the 8 experiments: the average maximum fitness (and standard deviation) of the 30 melodies, the highest fitness and the average target similarity of the new metric.

Interestingly, only the first experiment (pitch and chromatic tone) resulted in any melodies with a perfect fitness of 1.0. Another observation was a decrease in fitness as more context was taken into account, i.e. higher level n-grams.

Answering the question posed in the introduction, the target piece was never reproduced. Of the eight experiments with increasing number of metrics, only

Table 5.1 Average maximum fitness (std. dev.), highest fitness and average metric similarity of the 30 most fit melodies from the eight experiments.

Experiment	Fitness	Highest fitness	Metric similarity
1. Pitch and chromatic tone	0.990 (0.007)	1.0	0.987, 0.992
2. Melodic interval	0.967 (0.012)	0.989	0.933
3. Melodic bigram	0.921 (0.022)	0.964	0.822
4. Melodic trigram	0.827 (0.046)	0.903	0.655
5. Rhythm	0.867 (0.029)	0.939	0.975
6. Rhythmic interval	0.871 (0.036)	0.937	0.941
7. Rhythmic bigram	0.874 (0.035)	0.946	0.883
8. Rhythmic trigram	0.857 (0.036)	0.937	0.811

the first resulted in melodies with perfect fitness scores. This was somewhat surprising, since the target melody was very simple and the evolved melodies fairly short.

The sub-sections that follow provide more detailed results and analysis of each experiment. Relevant metric frequency distributions from the evolved music are included. When sensible, these plots show only the N most frequent elements when there is a lot of noise present. This is done so that they are easier to compare to their respective target distributions (Figure 5.2). An example melody is also given for each experiment, which helps illustrate how the melodies change with each additional metric.

1. Pitch and Chromatic Tone

As mentioned, the hypothesis was that pitch and chromatic tone would confine music to the scale and pitch region of the target piece. In other words, the evolved melodies were expected to have pitches between 60 and 72 and be in the C major scale.

As can be seen from the first row of Table 5.1, the 30 evolved melodies had very high fitness, indicating a close similarity to the target vector. Of the 30 melodies, nine had a perfect fitness score of 1.0, i.e. they had both the same pitches and frequencies as the target piece.

Figure 5.4 shows the pitch and chromatic tone distributions of the 30 evolved melodies. Compared to the target distributions (Figure 5.2), they are quite

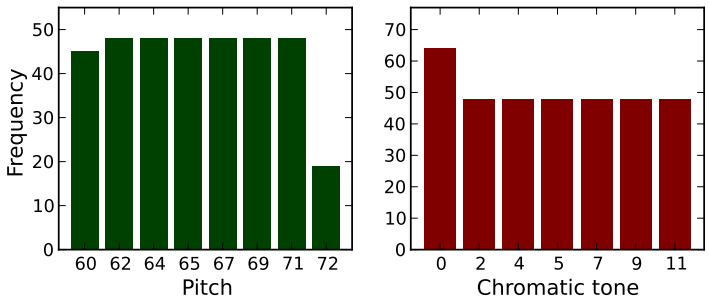


Figure 5.4 Frequency of pitch and chromatic tones in the 30 evolved melodies from Experiment 1.

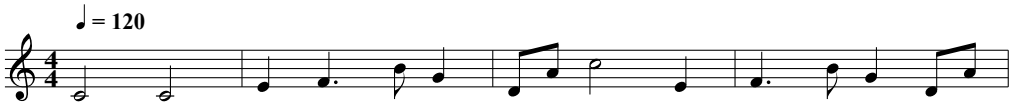


Figure 5.5 Example melody evolved with pitch and chromatic tone from Experiment 1. It has a perfect fitness of 1.0.

similar in shape. As can be seen, all of the evolved melodies used the same scale as the target – no pitches were outside of the pitch region or the scale. This confirmed the initial hypothesis.

However, not all the melodies contained *all* of the pitches found in the target piece. For instance, about one third of them did not include the highest pitch (72). Although displaying most of the correct pitches, 21 of the melodies did not use them at the target frequencies. In these cases, evolution was stuck in a local maximum.

Finally, listening to a few of the evolved melodies confirmed that the pitches appeared in seemingly random order, as was expected. Figure 5.5 shows one of the 9 melodies with perfect fitness.

2. Melodic Interval

When the melodic interval metric was added, it was hypothesized that it would ensure melodic locality, i.e. small changes in pitch. With the particular target piece, it was expected to primarily see the major (± 2) and minor second (± 1) intervals employed in the evolved melodies.

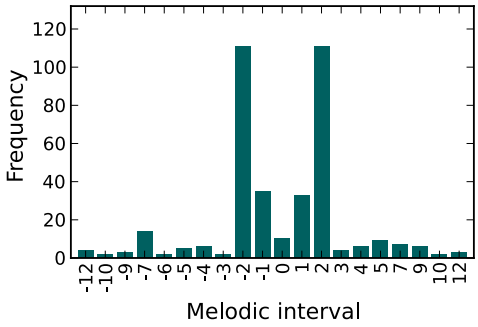


Figure 5.6 Frequency of melodic intervals in the evolved melodies from Experiment 2.

Rather surprisingly, none of the 30 evolutionary runs resulted in melodies with perfect fitness (1.0). The average metric similarity was only 0.93, demonstrating that melodic intervals are harder to satisfy since they include note *context*.

The frequency of all melodic intervals in the evolved melodies is depicted in Figure 5.6. Compared to the target melodic intervals in Figure 5.2, a similarity in shape can be found in the centre. That is, the intervals most frequently used in the evolved melodies are the major and minor second – just as in the target piece. This confirmed the hypothesis.

The evolved melodies contained a total of 23 different melodic intervals, i.e. a fair amount of noise compared to the four intervals in the target distribution. There is also a deviation in the relative frequency of the minor and major second: f_1/f_2. This ratio is $2/5 = 0.4$ for the target and approximately $35/111 \approx 0.32$ for the evolved music. So although the intervals were fairly similar to the target, the evolved melodies did have a few discrepancies.

Listening to a selection of the evolved pieces revealed a much more local melodic progression. Some melodies were even kind of pleasant, albeit with a few strange intervals and a random rhythm. Figure 5.7 shows the melody with the highest fitness. Note the small changes in pitch between notes.

3. Melodic Bigram

With the addition of melodic bigrams, it was expected to see shorter melodic motifs in the evolved music. The target piece consists of the C major scale

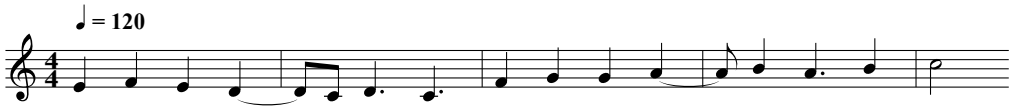


Figure 5.7 Evolved melody with the highest fitness when melodic intervals were enforced by fitness (Experiment 2). Note the small changes in pitch between notes.

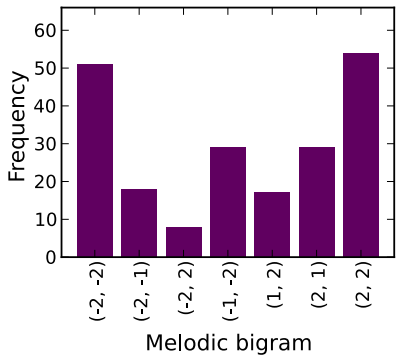


Figure 5.8 The seven most frequent melodic bigrams in the evolved music from Experiment 3.

played up and down. It was therefore predicted that the evolved melodies would show similar tendencies. How similar, however, would remain to see.

From Table 5.1 a decrease in fitness may be seen when melodic bigrams were introduced. The average melodic bigram similarity was 0.82, which again demonstrates that as more melodic context is taken into account, optimization becomes increasingly difficult for the EA.

Figure 5.8 depicts the seven most frequent melodic bigrams in the evolved music. In total, the 30 melodies contained 48 different bigrams – a considerable amount of noise. There is, however, a similarity between the the evolved distribution and the target. The six most frequent bigrams in the evolved music are all in the target distribution. Their relative frequencies are also somewhat similar, especially for the top four bigrams.

Listening to some of the evolved melodies revealed a clear similarity to the target piece. Although none were identical, they were consistently playing variations of the C major scale up and down (or vice versa). Figure 5.9 shows the melody with the highest fitness (0.96). When compared with the best melody from the previous experiment (Figure 5.7), it becomes clear that bigrams add a significant amount of knowledge to the fitness function.

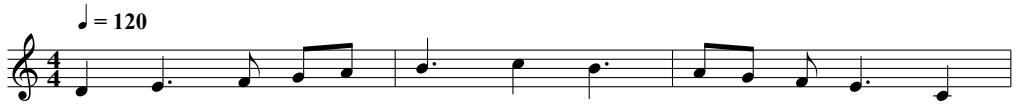


Figure 5.9 The melody with the highest fitness when melodic bigrams were included in Experiment 3. Notice the similarity to the target piece in Figure 5.1.

4. Melodic Trigram

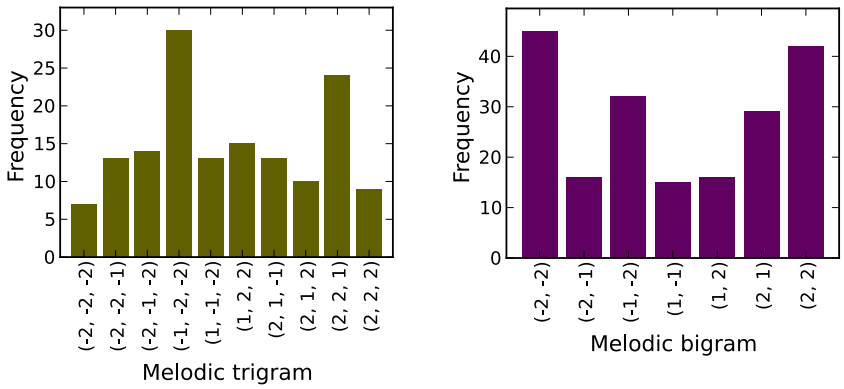
It was hypothesized that including melodic trigrams would promote melodic motifs similar to those found in the target. As a consequence, the evolved music was expected to become even more similar to the target piece. However, the previous experiment revealed that melodic bigrams resulted in melodies which were already very similar. It was therefore doubtful that the inclusion of trigrams would increase similarity at this point.

When melodic trigrams were added, there was a drop in fitness of approximately 0.1 (Table 5.1). The average metric similarity was only 0.66.

A total of 30 melodic trigrams were found in the evolved music, which is less noise compared to the bigrams in the previous experiment. In fact, the number of bigrams had *decreased* to 20. This was probably due to the fact that trigrams are composed of two adjacent bigrams – effectively doubling the fitness reward for correct bigrams. In other words, a higher fitness reward would be gained for correct bigrams than other metrics. In fact, the average bigram similarity had increased to 0.84 from the previous 0.82.

Figure 5.10a shows the 10 most frequent trigrams. As can be seen, all are indeed in the target distribution. An agreement in shape between the evolved and target distributions can also be observed. The relative frequencies, however, are deviating by a fair amount. In addition, Figure 5.10b depicts the bigram frequencies. It is easy to see that this distribution is more similar to the target, when compared to Figure 5.8.

There seemed to be an improvement in bigram similarity, but there was a decrease in the overall fitness. Consequently it was expected to see a decrease in some of the other metric similarities. This indeed proved to be the case – the average pitch and chromatic tone similarity was only 0.84 and 0.86, respectively. It seemed that evolution sacrificed a correct *scale* for similarity in the melodic bigrams and trigrams. Whereas almost all of the melodies up until now had followed the C major scale, ten pieces were now found to use notes outside of the scale.



(a) **(b)**

Figure 5.10 The 10 most frequent melodic trigrams (a) and 7 most frequent melodic bigrams (b) in the evolved music from Experiment 4, where melodic trigram similarity was included in the fitness function.

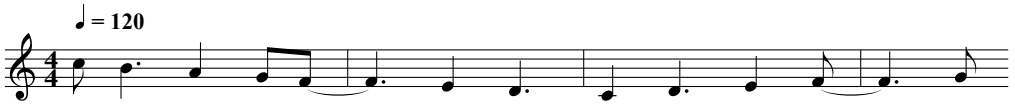


Figure 5.11 The most fit melody from Experiment 4, where melodic trigrams were introduced.

Listening to the top melodies revealed no clear improvements. At least when such a simple and short melody was used as the target, melodic bigrams seemed to contain sufficient knowledge alone. In Figure 5.11, the most fit melody can be seen. Notice how it resembles the target melody inverted (turned upside-down).

5. Rhythm

So far, the evolved melodies displayed a rather random, inconsistent rhythm. This came as no surprise, as none of the metrics had taken note durations into account. Recall from Section 3.2.1 that the rhythm metric measures the frequency of durations (since rests are not allowed in the genome). It was estimated that rhythm would enforce a tempo similar to the target piece.

An overall increase in fitness can be seen in Table 5.1, which is due to the high metric similarity of rhythm (0.975). As such, rhythm seemed to be a

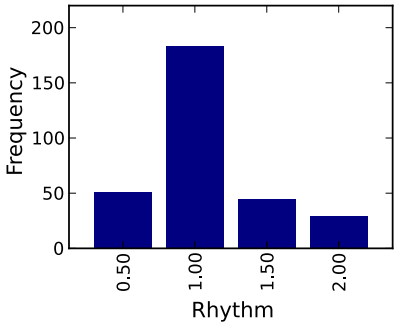


Figure 5.12 Frequency of rhythm (note durations) in the evolved music from Experiment 5. Note that durations are in quarter notes.

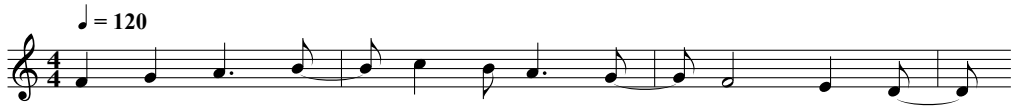


Figure 5.13 Melody with the highest fitness after rhythm was added to the fitness function in Experiment 5.

fairly easy objective. However, none of the runs resulted in perfect rhythm similarity. This was surprising, given that none of the other metrics operate on durations, i.e. there shouldn't be any conflicts of interest. It was suspected that the other metrics were given priority (being the majority) and that the genetic operators were unable to fine-tune the duration of fit individuals. This was partly confirmed by running the experiment with the rhythm metric alone, which resulted in perfect similarity for all the melodies.

Figure 5.12 shows the rhythm frequency distribution of the evolved music. Since durations were limited to the four found in the target piece, no noise was present. Compared to the target distribution in Figure 5.2, there is a clear similarity. Note that in the figures, durations are in multiples of the quarter note, e.g. 0.5 is the eight note, 1.0 is the quarter note, 1.5 the augmented quarter note (3/8) and 2.0 denotes the half note.

Listening to the most fit melodies indeed revealed a tempo more similar to the target piece. Figure 5.13 shows the melody with the highest fitness. As can be seen, it contains mostly quarter notes, as with the target melody. The rhythm, however, is rather free (random) and somewhat inconsistent.

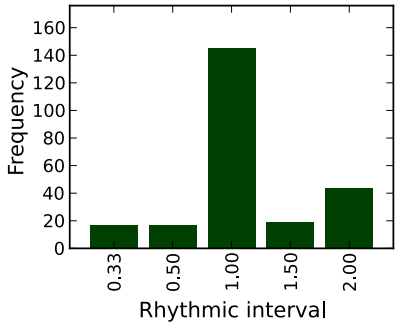


Figure 5.14 Five most frequent rhythmic intervals in the evolved music from Experiment 6. Note that durations are in quarter notes.

6. Rhythmic Interval

It was believed that the addition of rhythmic intervals would improve the rhythmic consistency, as it measures the relationship between the duration of adjacent notes. As mentioned, it was expected to see some pattern in which notes follow each other, e.g. augmented quarter notes (1.5) would be followed by eight notes (0.5), eight notes followed by quarter notes (1.0) and so on, similar to the target piece.

As can be seen in Table 5.1, there was a high similarity of rhythmic intervals in the evolved music (0.94). A total of 11 rhythmic intervals were found in the evolved melodies, so some noise was present. Figure 5.14 shows the five most frequent intervals. Although fairly similar to the target, the EA seemed to struggle with the particular interval 0.33. This was likely due to the fact that there is only one combination of the four note durations that results in this particular interval – namely $(1.5, 0.5)$. Another frequent interval was 0.5, which is not found in the target distribution.

After listening to the most fit melodies, it was easy to recognize several rhythmic features derived from the target melody. Figure 5.15 shows the melody with the highest fitness. Notice the familiar rhythmic motif at the beginning of the melody – an augmented quarter note, followed by an eight note and then a quarter note.

7. Rhythmic Bigram

The hypothesis was that rhythmic bigrams would promote short rhythmic motifs. One such characteristic motif can be found in the first and third bar

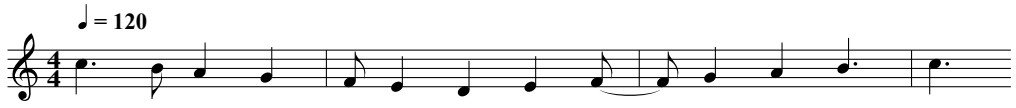


Figure 5.15 The most fit melody evolved with rhythmic intervals from Experiment 6. Notice the familiar rhythmic motif at the beginning.

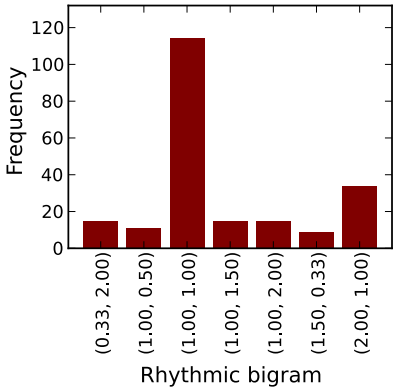


Figure 5.16 The seven most frequent rhythmic bigrams in the evolved music from Experiment 7. Note that durations are in quarter notes.

of the target music (5.1) – the note sequence ♩.♪♩ which corresponds to the rhythmic bigram $(0.33, 2.0)$.

Addition of rhythmic bigrams resulted in fitness values fairly similar to the previous experiment. The average rhythmic bigram similarity was 0.88.

In total, 31 rhythmic bigrams were found in the evolved music, i.e. a fair amount of noise compared to the six bigrams in the target distribution. Figure 5.16 shows the seven most frequently used bigrams. As can be seen, the evolved melodies were dominated by the bigrams $(1.0, 1.0)$ and $(2.0, 1.0)$, similar to the target (Figure 5.2). Although many of the other bigrams from the target are present in the distribution, their relative frequencies are deviating considerably. The bigram $(0.33, 2.0)$, for instance, does not occur nearly as often as in the target distribution.

Listening to the most fit melodies revealed a more consistent rhythm similar to the target piece. The characteristic rhythm corresponding to the $(0.33, 2.0)$ bigram, however, was not as frequent as predicted. This is perhaps not so surprising, since the EA was struggling with the 0.33 interval in the previous experiment. Figure 5.17 shows the melody with the highest fitness. Notice how the characteristic rhythmic motif $(0.33, 2.0)$ is *not* present.

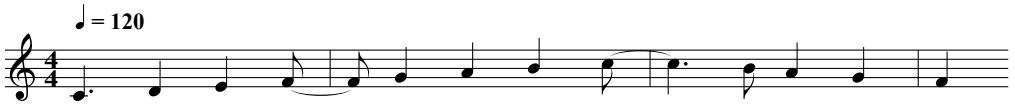


Figure 5.17 Melody with highest fitness from Experiment 7, evolved with rhythmic bigrams.

8. Rhythmic Trigram

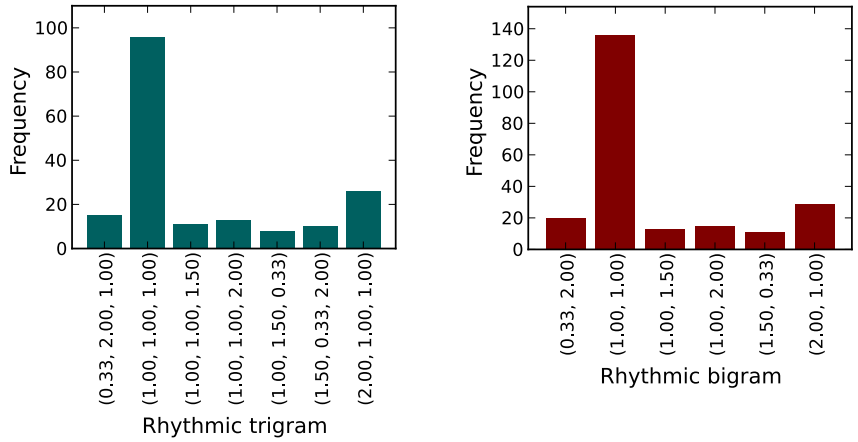
Rhythmic trigrams were believed to promote longer rhythmic motifs. It was thus anticipated to see more rhythmic similarity between the evolved music and the target piece.

As observed in Table 5.1, the average rhythmic trigram similarity was 0.81 and a slight decrease in fitness was seen when rhythmic trigrams were included.

The evolved music contained a total of 43 different rhythmic trigrams – a fair amount of noise. As with the melodic trigrams in Experiment 4, the number of rhythmic bigrams had decreased to 25 with the inclusion of rhythmic trigrams. There was also an increase in average rhythmic bigram similarity – 0.92 compared to 0.88 from the previous experiment. As seen in Figure 5.18a, the seven most frequent trigrams in the evolved music were the same trigrams as in the target distribution. The relative frequencies are however not as similar, with a clear bias towards $(1.0, 1.0, 1.0)$. In the target distribution, the $(0.33, 2.0, 1.0)$ trigram is 50% as frequent as $(1.0, 1.0, 1.0)$. However, for the evolved music the equivalent number is only 16%.

Figure 5.18b shows the frequency of rhythmic bigrams from this experiment (top 6). Note how the frequency of the bigram $(0.33, 2.0)$ has increased slightly compared to the previous results (Section 5.1.4). It is, however, nowhere near the target frequency.

After listening to the most fit melodies, there were no clear rhythmic improvements. As the rhythmic trigram similarity was fairly low (0.81), this was perhaps not so surprising. Figure 5.19 shows the melody with the highest fitness. It illustrates an important limitation to the rhythm metrics: notes are free to span across bars, which can often result in a strange rhythm. Notice how the target melody contains no such bar-spanning notes. Since *time signature* is not captured by the rhythm metrics, this feature is lost.



(a) **(b)**

Figure 5.18 The seven most frequent rhythmic trigrams (a) and top six rhythmic bigrams (b) in the evolved music from Experiment 8, where rhythmic trigrams were included in the fitness function. Note that durations are in quarter notes.

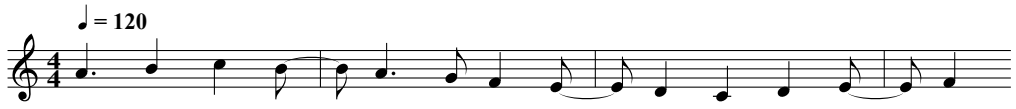


Figure 5.19 The melody with the highest fitness from Experiment 8, evolved with rhythmic trigrams included.

5.1.5 Summary

Experiment 5.1 demonstrated the usefulness of the eight metrics explored. By using a simple target melody, it was possible to study the effects of each metric on the evolved music. An incremental approach was taken: the set of metrics used for fitness was gradually expanded by adding one metric at a time and analysing the results.

The pitch and chromatic tone metrics successfully confined the music to the scale and pitch region of the target piece. Melodic locality was ensured by the melodic interval metrics, which resulted in some rather pleasant melodies. Melodic bigrams and trigrams were found to promote motifs. Results showed that bigrams were sufficient to evolve melody-wise very similar pieces. The addition of melodic trigrams resulted in higher bigram similarity, but no clear improvement in the music was seen.

For the evolutionary algorithm, the rhythm-based metrics seemed to be a tough nut to crack. The rhythm metric enforced a tempo similar to that of the target. Rhythmic consistency was improved with the addition of rhythmic intervals, with the emergence of several rhythmic motifs similar to those in the target. However, some intervals proved to be problematic, which reduced the similarity as rhythmic bigrams and trigrams were introduced. This resulted in considerably fewer rhythmic motifs than first predicted.

It seemed to be problematic for the EA to optimize several metrics at the same time. A recurring observation was that the addition of a new metric resulted in the decrease in similarity for the other metrics. This was especially true for conflicting metrics, such as pitch and melodic interval, where the increasing similarity of one often results in a decreased similarity in the other.

The EA failed to find any melodies with perfect similarity scores for rhythm in Experiment 5. This was unexpected because, at that point, none of the other metrics were operating on note durations, i.e. there were no conflicting metrics. It was suspected that the EA was unable to fine-tune melodies which already had high scores related to the pitch-based metrics.

5.2 Improving the Basics

The results of the previous experiment demonstrated the effects of each metric on the evolved music. There were, however, some problems when using many metrics at the same time for fitness. In particular, the EA seemed to struggle with the metrics related to rhythm. It was suspected that mutation was unable to fine-tune single notes and was the main cause for the failure. In an attempt to improve the fitness, the function and terminal probabilities were changed to make mutation more fine-grained.

5.2.1 Introduction

As mentioned, the hypothesis was that mutation was changing too many notes at the same time. With high fitness individuals, this will in most cases result in a fitness decrease. In other words, mutation was *destructive* in these cases.

The first evidence of this phenomenon was found when the rhythm metric was added (Section 5.1.4). At this point, it was the *only* metric operating on note durations and a simple one at that. It was therefore expected that achieving a near perfect rhythm similarity would be an easy task for the EA. However, this proved not to be the case, with the average rhythm similarity at 0.97 and no melodies with a perfect score of 1.0. The higher level rhythm metrics (intervals and n-grams) also seemed to be problematic, with rather poor similarity between the evolved frequency distributions and the targets.

As described in Section 3.1, the mutation operator selects a random node and replaces it with a randomly generated tree. What type of node is selected, is determined by the function and terminal probability parameters. By default, these were set to 0.9 and 0.1, respectively, which causes mutation to choose a function node 90% of the time and terminal node 10% of the time. It was thus much more likely for mutation (and crossover) to affect many notes, than it was to change a single note.

It was therefore decided to try different values for these two parameters to see if improvements in fitness could be achieved. In particular, the goal was to increase the similarity of the rhythm-based metrics, something which would hopefully result in improved rhythmic characteristics of the evolved music.

5.2.2 Setup

Experiment 8 from Section 5.1, i.e. using all nine metrics, was re-run with the following function and terminal probabilities:

1. Both function and terminal probability at 0.5
2. Function and terminal probability at 0.1 and 0.9, respectively

The setup was otherwise the same as before (see Section 5.1.2). Evolution was run for 500 generations, and the average maximum fitness was plotted over 30 runs.

5.2.3 Results and Discussion

Figure 5.20 shows the fitness plots from the two configurations of function and terminal probabilities. The fitness from the previous experiment is also included, where the two parameters were at their default values. As can be seen, there were *significant* improvement in fitness when the terminal probability was increased. The best performer was configuration 2, i.e. a very high terminal probability.

In Table 5.2, the average metric similarities from the previous experiment (old) and configuration 2 are listed, as well as the relative fitness improvement (difference). The overall fitness was improved considerably: from 0.857 to 0.911. As seen, similarity increased across all metrics. The metric with the most change is rhythmic trigram, with an increased similarity of 0.106. It is closely followed by melodic trigram, which had a growth of 0.087.

As mentioned, the main goal was to increase the rhythmic similarity of the evolved music, something which would hopefully yield better rhythmic qualities in the melodies. Examination of the evolved frequency distributions revealed a general trend: there was a higher frequency of correct events and consequently less noise. For instance, the evolved music now contained 28 different rhythmic trigrams, as opposed to 43 as previously seen (Section 5.1.4).

The rhythmic interval 0.33 had previously been problematic, with only 22 occurrences in the evolved music. Now it was much more frequent, appearing

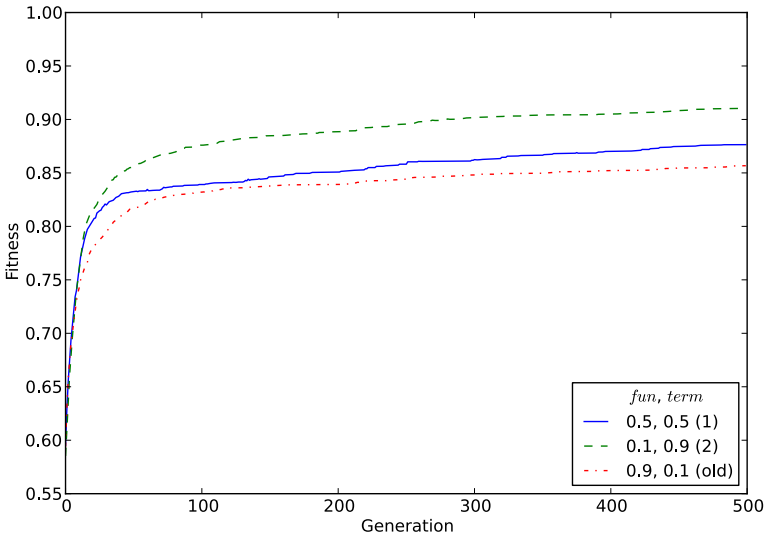


Figure 5.20 Fitness plots for the three different values of function and terminal probabilities. As can be seen, a terminal probability of 0.9 resulted in the highest fitness.

Table 5.2 The average metric similarities when using function and terminal probabilities of respectively: 0.9 and 0.1 (old); 0.1 and 0.9 (2).

Metric	0.9 and 0.1 (old)	0.1 and 0.9 (2)	Difference
Pitch	0.846	0.903	0.057
Chromatic tone	0.857	0.917	0.060
Melodic interval	0.952	0.963	0.012
Melodic bigram	0.823	0.874	0.052
Melodic trigram	0.595	0.682	0.087
Rhythm	0.946	0.991	0.045
Rhythmic interval	0.962	0.984	0.022
Rhythmic bigram	0.919	0.965	0.046
Rhythmic trigram	0.811	0.917	0.106
Fitness	0.857	0.911	0.054

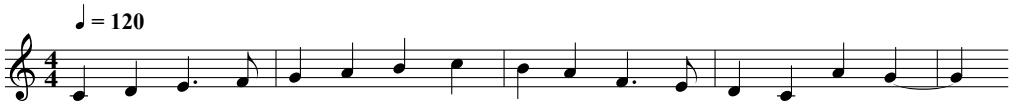


Figure 5.21 Evolved melody which exhibits *two* instances of the characteristic rhythmic motif $(0.33, 2.0, 1.0)$: ♩.♪♩♩

41 times. Similar improvements were found for rhythmic bigrams and trigrams. The characteristic rhythmic motif $(0.33, 2.0, 1.0)$ had a frequency of 35 as opposed to 15 as previously seen. In other words, most of the evolved melodies now had at least one instance of this rhythmic motif. This was a satisfying improvement.

Figure 5.21 shows one of the evolved melodies which exhibits *two* instances of the characteristic rhythmic motif. It has a total fitness of 0.933 and high rhythmic similarities: 1.0, 0.994, 0.968 and 0.946 for rhythm, rhythmic intervals, rhythmic bigrams and rhythmic trigrams, respectively.

As with any computer system, there were still room for improvements. For instance, adjusting the weights of each metric might help increase overall fitness. Other types of function nodes in the representation might also yield benefits, e.g. based on musical permutations such as transposition and inversion. Another possibility could be to evolve melody and rhythm separately.

However, at this point it was concluded that the results were good enough. It was time to expose the system to some real-world music.

5.2.4 Summary

Increasing the terminal probability from 0.1 to 0.9 greatly improved the overall fitness of the evolved melodies. This allowed the EA to more easily fine-tune individual notes. In general, the evolved melodies had a higher frequency of correct musical events and as a result less noise.

The main motivation for improving fitness was to increase the rhythmic similarity of the evolved music. The results showed a significant improvement for the rhythmic metrics, especially the rhythmic trigram. Further examination revealed a major increase in the frequency of the characteristic rhythmic motif derived from the target melody.

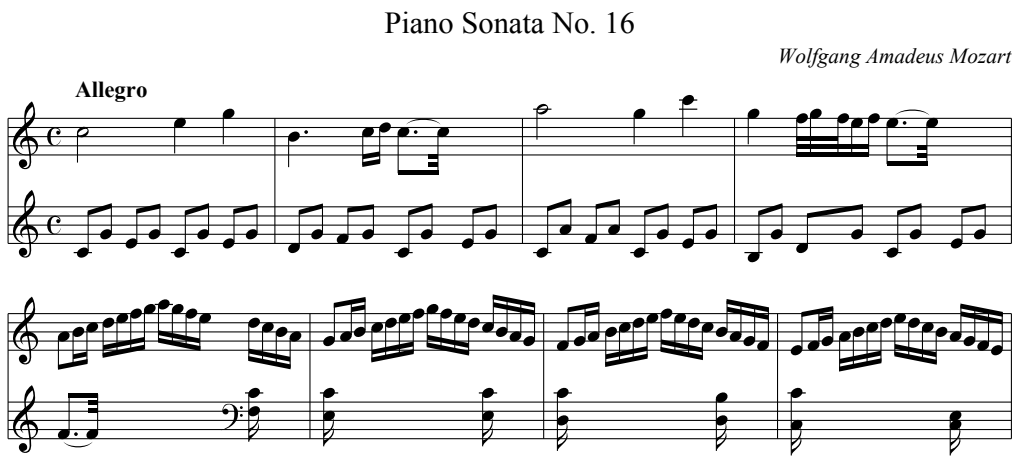


Figure 5.22 The first eight bars of the first target piece: *Mozart's Piano Sonata No. 16 in C major (K. 545).*

5.3 Learning From the Best

So far, the similarity-based fitness function had been applied to a very simple target melody, which was constructed specifically for testing purposes. In this experiment, two real-world music pieces were used as targets. The artists excel at quite different musical styles and lived in different centuries: *Mozart* and *The Beatles*. The goal was to examine the fitness function's ability to capture the *musical style* of the pieces.

5.3.1 Introduction

Since the goal was to see how well the fitness function could model musical style, two target pieces were selected that are easily distinguishable and have several characteristic features.

The first target piece is *Mozart's Piano Sonata No. 16 in C major (K. 545)*, first movement (Allegro), composed in 1788. The reader is likely familiar with the melody – the first eight bars are shown in Figure 5.22. It can be characterized as having ascending and descending melodic motions in a quick tempo.

The Beatles' Let it Be was used as the second target piece. The hit pop-song from 1970 features a slow tempo and minimalist melody. Figure 5.23 shows the first four bars of the song.

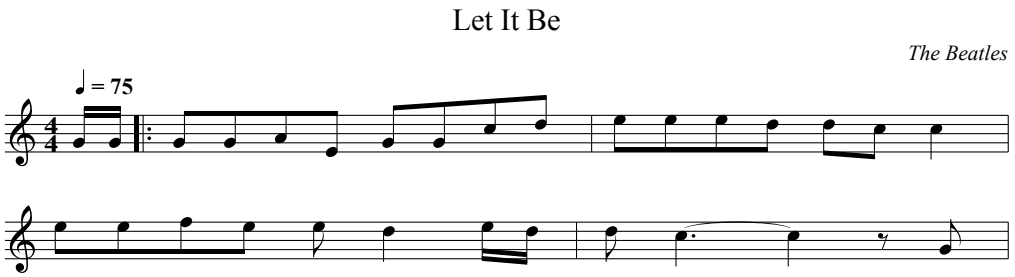


Figure 5.23 First four bars of the second target piece: *The Beatles' Let It Be.*

5.3.2 Setup

The setup was similar to Experiment 5.1: For each target piece, evolution was run for 500 generations and 30 runs were carried out. As before, the most fit melody at the end of each run was kept for analysis.

The parameters are listed below:

- Population size: 100
- Mutation rate: 0.1
- Crossover rate: 0.9
- Tournament selection: $k = 5$, $e = 0.1$
- Fitness: Cosine similarity to target piece
  - Metrics: pitch, chromatic tone, melodic interval, melodic bigram, melodic trigram, rhythm, rhythmic interval, rhythmic bigram, rhythmic trigram
  - Target vectors: *experiment dependent*
  - Filtering of target vectors: 1%
  - Weights: uniform (1.0)

- Representation: Tree
  - Functions: concatenation (+)
  - Terminals: *simple* (pitch, duration)
  - Initialization method: *full*
  - Max tree depth: 6
  - Pitches: *experiment dependent*
  - Pitch reference: *experiment dependent*
  - Durations: *experiment dependent*
  - Scale: chromatic (no scale)
  - Function probability: 0.1
  - Terminal probability: 0.9

The maximum tree depth was set to 6, allowing for a maximum of 64 notes, granting the EA more freedom than in the previous experiments. The function and terminal probability were based on the results from Experiment 5.2.

5.3.3 Experiment Setup

For each target piece the melody was identified and analysed by the nine metrics, producing a set of target vectors that were used in the fitness function. The number of pitches, pitch reference and possible durations were derived from each target piece, as described in Section 5.1.2.

Mozart

The genotype and phenotype parameters derived from Mozart's piece were:

- Pitches: 25
- Pitch reference: 62 (D)
- Duration map: $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{5}{16}$, $\frac{1}{2}$, $\frac{9}{16}$ and $\frac{3}{4}$

Figure 5.24 shows the target frequency distributions from Mozart's Piano Sonata. Filtering resulted in the removal of 405 data points below the 1% threshold. Most notably, there were 167 melodic trigrams below the threshold, which gives an indication of the melodic complexity of the piece.

Evidence of the ascending and descending melody (see Figure 5.22) can be seen in the melodic interval distribution, where the major (± 2) and minor (± 1) second have high frequencies.

The sonata exhibits a fast tempo, which is apparent in the rhythm distribution where the sixteenth note (0.25) is dominant. There is also little rhythmic variation, evident from the clear majority of the rhythmic interval 1.0.

The Beatles

For *Let it Be*, the derived genotype and phenotype parameters were:

- Pitches: 18
- Pitch reference: 64 (E)
- Durations: $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{5}{8}$ and $\frac{3}{4}$.

In Figure 5.25, the target distributions from The Beatles' hit are depicted. In contrast to Mozart's piece, filtering only reduced the number of events by 5 due to the flatter bigram and trigram distributions.

Notice how the song contains only 9 different pitches, dominated by the G (67), C (72), D (74) and E (76). The melody displays a fair amount of same-note repetition, apparent from the melodic intervals where the unison (0) has the highest frequency.

The eight note (0.5) is the duration most frequently used in the piece. Some rhythmic variation is also found, illustrated by the diversity of rhythmic intervals, bigrams and trigrams. Repetition of the same note duration, i.e. rhythmic interval 1.0, is in clear majority though.

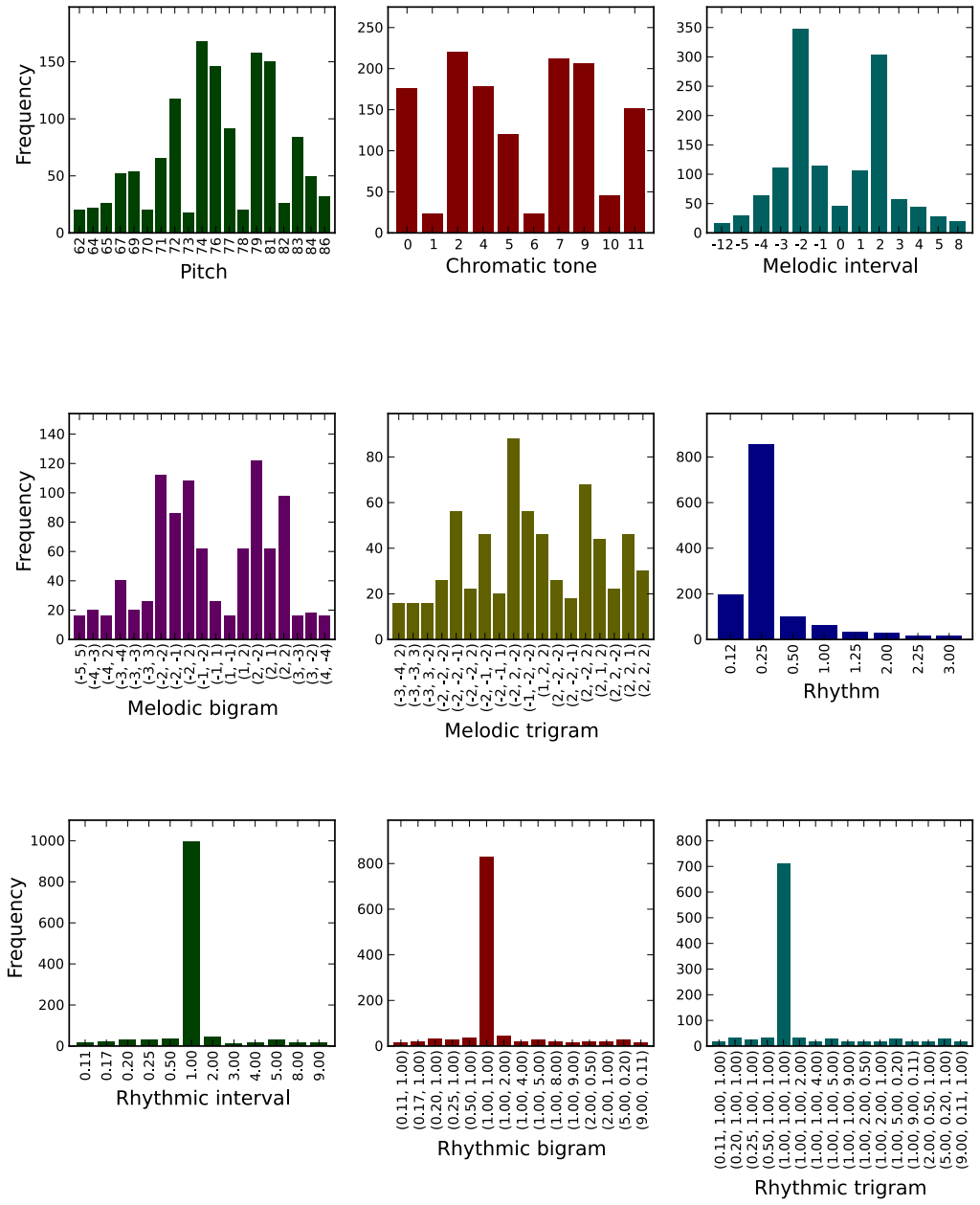


Figure 5.24 Frequency of musical events in *Mozart's Piano Sonata No. 16 in C major*. Events that account for less than 1% of the distribution have been filtered out.

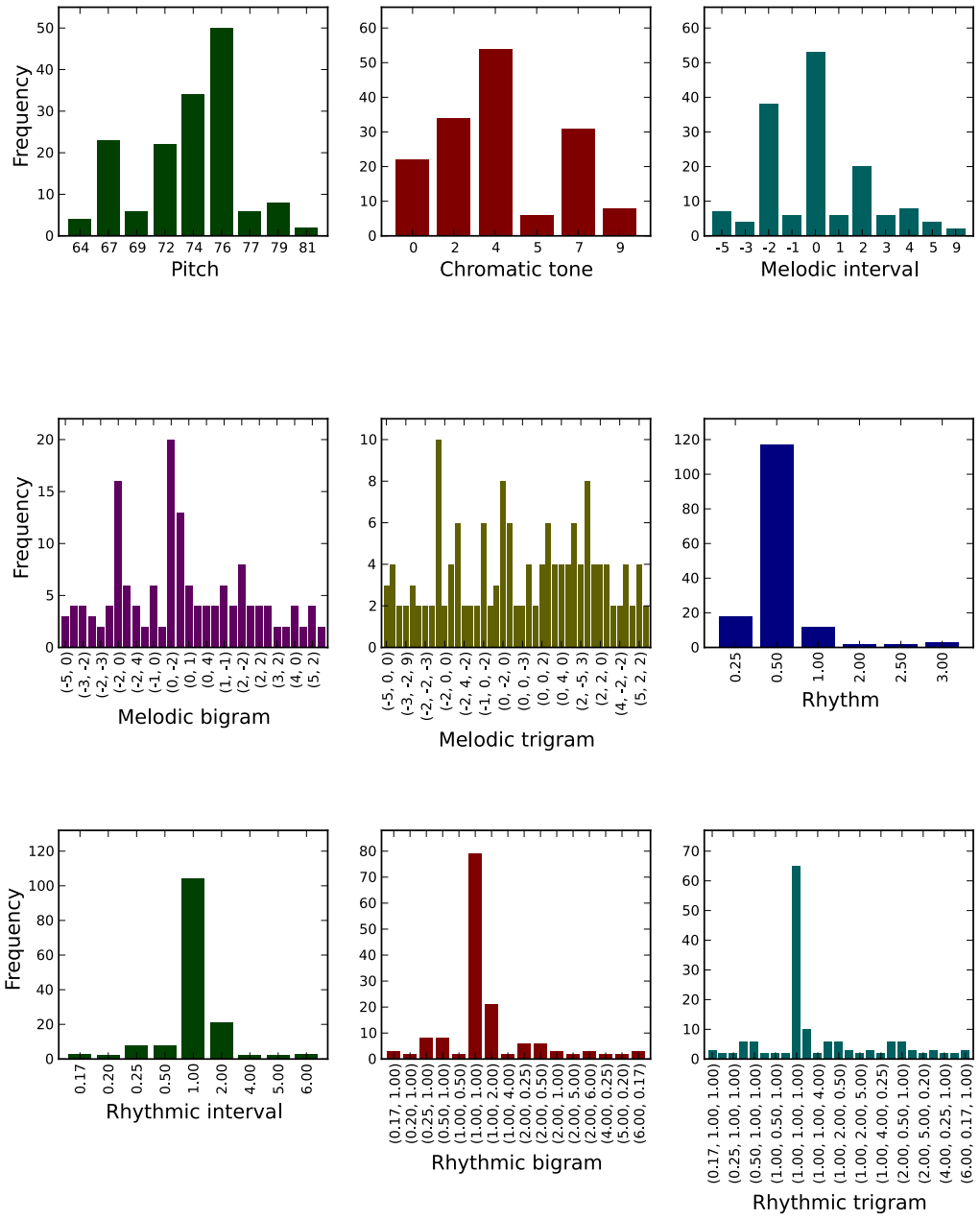


Figure 5.25 Frequency of different types of musical events in *The Beatles' Let it Be*. Events with relative frequencies of less than 1% have been filtered out. For the sake of readability, some values have been left out in the denser plots.

Table 5.3 Melody statistics from the two experiments using different target pieces: average maximum fitness (std. dev.), highest fitness and the average number of notes.

Target Piece	Fitness	Highest fitness	Number of notes
Mozart	0.949 (0.012)	0.972	46.87
The Beatles	0.960 (0.007)	0.972	45.63

5.3.4 Results and Discussion

Table 5.3 lists some interesting statistics from the two experiments. The average fitness was consistently high for both target pieces. In general, the music evolved with The Beatles' song reached slightly higher fitness than Mozart's Piano Sonata, but the most fit melody from both sets had the same fitness score. The number of notes were also approximately the same.

A higher variation in fitness was seen within the Mozart-melodies, where the lowest fitness of the 30 melodies was 0.918. The equivalent number from The Beatles-melodies was 0.945. As such, the target vectors from Mozart's piece provided a more difficult optimisation task.

Mozart

After listening to the melodies evolved with *Mozart's Piano Sonata No. 16* as target, it was clear that the 30 melodies were quite similar to each other. As such, the fitness function seemed to produce fairly *consistent* results. Figure 5.26 shows a typical example.

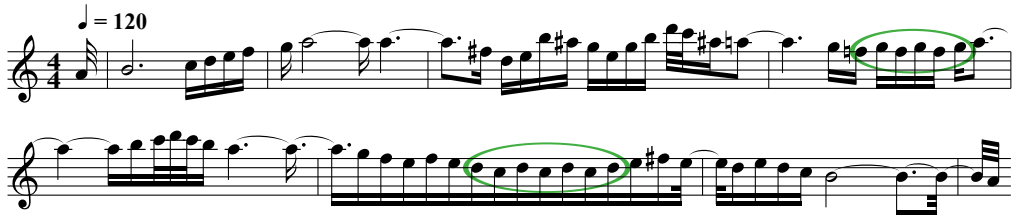


Figure 5.26 Typical example melody evolved with *Mozart's Piano Sonata No. 16* as inspirational target.

As can be seen, the music exhibits a fast tempo matching Mozart's piece. The melodic features are also periodically similar, with several ascending and descending motions. These motions were, however, generally not as long as in the original piece.

Another typical trait was alternation between two adjacent notes (circled in the figure). These were likely due to the many *trills* found in the sonata, which are *rapid* alternations between two adjacent notes. However, such "trills" were also used with a *slow* tempo, which resulted in very monotone phrases. In other words, there seemed to be no correspondence between rhythm and melody. This uncovered an important limitation of the metrics used: no correlation between pitch and duration is captured, since they all operate exclusively on one or the other. It is hypothesized that the pitch duration metric might help in this area.

Rhythmically, the melodies were perceived as very random and inconsistent. When listening, there was no clear concept of a *beat* – the rhythm was floating rather freely. Mozart's piece, in contrast, displays a strict adherence to the time signature. Metrics which take time signature into account might improve the results in this aspect.

Another issue was seemingly random changes in tempo within melodies. Typical was the sudden change in note durations, which often occurred several times within each short melody. This was likely due to the fact that the piano sonata consists of several themes, some of which are slower than the others. Given that the evolved melodies were much shorter than Mozart's piece, these themes were necessarily compressed in order to satisfy the target distributions. In other words, the metrics were unable to capture the *global* structure of Mozart's piece.

Generally, the evolved pieces were rather *unpleasant*, with the random rhythm emerging as the major issue. The Piano Sonata was simply too complex for the metrics, resulting in rather chaotic melodies. At a local level, however, it was clear that the music was Mozart-inspired.

The Beatles

The melodies evolved with *Let it Be* applied as target were also strikingly similar to each other. However, compared to the Mozart results they were much simpler. A typical example can be seen in Figure 5.27.

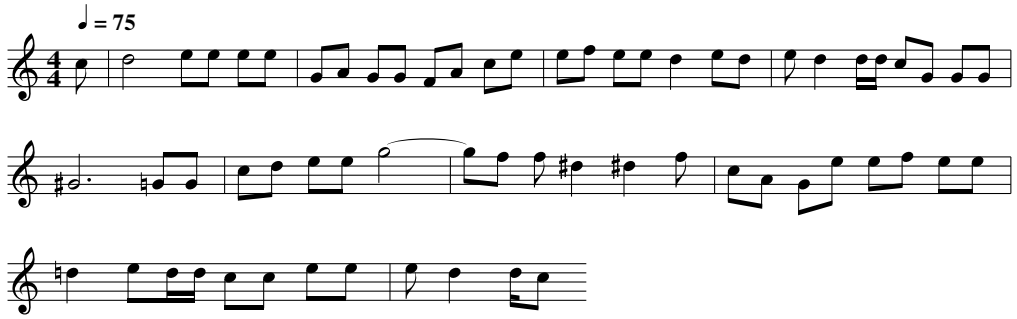


Figure 5.27 Typical melody evolved with *The Beatles' Let it Be* applied as target.

Like with The Beatles' song, the evolved melodies featured a slow tempo and were also quite minimalistic.

Many melodic motifs were easily recognized as derivatives from *Let it Be*. At times though, the melody was perceived as *too* similar – small excerpts from *Let it Be* would appear in between otherwise unfamiliar motifs. In these cases the listener's expectations were not fulfilled.

Unfortunately, the rhythm was often rather strange and inconsistent, with no clear beat or time signature. There were also no apparent coherence between rhythm and melody. Similar issues were also discovered in the Mozart-melodies, as discussed earlier. Since *Let it Be* features a fairly uniform speed, there were no sudden changes in tempo, which resulted in a more coherent musical structure than what was found in the Mozart-melodies.

Despite the problems outlined above, many of the melodies were in fact quite *pleasant*. This was an improvement over the melodies evolved using Mozart as target, which were mostly perceived as noisy and unpleasant. In other words, the similarity metrics were more successful when the simpler melody was applied as target.

5.3.5 Summary

The two experiments demonstrated that melodies with high fitness could be evolved with two quite different real-world music pieces applied as targets. Within each experiment, the evolved melodies were very similar to each other.

The first experiment created music that was, at least locally, somewhat similar to the target Mozart piece. However, Mozart's Piano Sonata was likely too complex for the metrics applied, and the melodies were generally unpleasant.

Greater success was achieved with The Beatles song applied as target, which is a lot simpler compared to Mozart. Quite a few of these melodies were perceived as rather pleasant.

The experiments highlighted several issues with the current set of metrics. The randomly floating rhythm was found to be the main source of unpleasantness. Another issue was no correlation between melody and rhythm.

Finally, from the Mozart results it became clear that the application of a long target melody for the evolution of short melodies was challenging. Global structure in the target piece was necessarily compressed into much shorter time intervals.

5.4 Conclusions

The goal of the first experiment was preliminary testing of the similarity-based fitness function. As such, a very simple target melody was applied, constructed specifically for testing purposes. Furthermore, metrics were added to the fitness function in an incremental fashion.

Analysis of the results revealed several interesting features in the evolved pieces. As more metrics were applied, the evolved melodies became more and more similar to the target piece, although identical melodies never emerged. Metrics related to pitch yielded satisfactory results, but the rhythm metrics proved to be problematic.

Issues with the genotype were also uncovered and performance was greatly improved during the second experiment.

The most interesting results emerged from the last experiment, where real-world music was applied as fitness. The high consistency observed within the results was a valuable feature that made analysis much easier, which is promising for future improvements.

Both target pieces resulted in melodies that exhibited similar musical properties, at least at a local level. Mozart's piece was likely too complex for the

metrics, because of their inability to capture higher level musical structure. Quite pleasant melodies were however evolved when The Beatles' song was applied as the target. The metrics seemed to perform better when applied to the simpler piece.

Rhythmically, the evolved melodies lacked important features. Most importantly they did not display any clear *beat*. Although the rhythm-metrics were shown to improve rhythmic consistency, none of them take the *time signature* into account. Future work should focus on improvements in this area by devising new metrics. Counting note *offsets* within each bar might be a viable technique.

Even though there were several weaknesses found in the evolved music, they were generally much more pleasant than what was evolved with the Zipf approach. For instance, the melodic intervals were significantly more musical thanks to the knowledge-rich similarity-based fitness function.

Chapter 6

Conclusion and Future Work

Evolutionary techniques have shown great potential for musical tasks. Still, a major challenge in Evolutionary Music Composition systems is the design of a fitness function. Because human music creation is such an elusive process that is not fully understood, formalizing music knowledge is particularly difficult.

As stated in the introduction, the research goal was to design an *automatic* fitness function which could capture music *pleasantness* and be applied to evolve *different styles* of music.

This thesis presents a *quantitative approach* to automatic fitness, applied for the evolution of novel monophonic melodies. Two fitness functions were devised, both of which operate on frequency distributions of music events. As such they both depend heavily on the set of *metrics* used to create the distributions.

The first fitness function based on Zipf's Law evaluates music solely based on *scaling* properties. For this approach, it was necessary with additional music knowledge which was implemented as constraints in the representation. Although results revealed that pleasant music could be evolved with such a technique, only a minority of the results were in fact perceived as pleasant. In other words, the method proved to be insufficient for musical fitness alone. Furthermore it was deemed far too knowledge-weak to capture musical style.

Based on this experience, a new fitness function was devised which incorporates more music knowledge. The fitness is based on the *similarity* between the metric frequency distributions of the evolved music and some inspirational *target piece*. In other words, music with similar statistical properties as the target piece is rewarded. The method evaluates both the scaling properties and *content* of the evolved music. Furthermore, music knowledge encoded in the representation was no longer necessary.

Application of the similarity-based fitness function greatly improved the quality of the evolved music. When real-world music pieces were used as targets, quite a few pleasant results emerged from the population. The results were also highly consistent, in contrast to the melodies evolved with Zipf-based fitness which displayed significant variation. This consistency made analysis much easier and is promising for future improvements.

Musical style was also captured to some extent by the similarity approach. At a local level, the evolved melodies exhibited several features that resembled those found in the target music. The technique proved to be most successful when a fairly short music piece was used as the target. The application of a longer and more complex target piece was more challenging and is unlikely to yield satisfactory results at this point.

Still, more work is needed to improve the method. As mentioned, the technique is highly dependant on the metrics that are employed. Additional metrics are necessary to incorporate other important musical properties. Of particular interest are metrics for rhythm, melody-rhythm coherence and higher level musical structure. It is unfortunately difficult to know which features are sensible in the musical domain. In some cases, an educated guess can be made, but often the only way to know is through trial and error.

A hybrid approach would also be interesting to explore, with both Zipf and similarity-based fitness applied on different metrics. The Zipf approach might be particularly useful for higher level features such as melodic n-grams. By looking only at the scaling properties in these features, plagiarism of the target piece would be reduced, thus improving originality. Lower level features like melodic intervals are likely to be more general and should thus be applied in a similarity-based manner.

Utilization of several target pieces at the same time would certainly also be of interest. Imagine the learning potential in a large corpus of music. The technique is however still in it's infancy and requires much work before it can be applied to large data sets.

Nevertheless, fitness based on statistical similarity is a promising technique for evolutionary music systems. Advantages include *learning* from existing music pieces and *scalability* to different musical styles.

Bibliography

Bellinger, E. (2011). Little Ludwig, an evolutionary learning machine for musical composition. *Journal of the ACM*, 1(1).

Bentley, P. and Corne, D. (2002). *Creative evolutionary systems*. Morgan Kaufmann, 1. edition.

Biles, J., Anderson, P., and Loggi, L. (1996). Neural network fitness functions for a musical IGA. In *International ICSC Symposium on Intelligent Industrial Automation and Soft Computing*. International Computing Sciences Conferences (ICSC).

Biles, J. A. (1994). GenJam : A Genetic Algorithm for Generating Jazz Solos. In *Proceedings of the International Computer Music Conference*.

Dahlstedt, P. (2007). Autonomous evolution of complete piano pieces and performances. In *Proceedings of Music AL Workshop*.

Duda, R. O., Hart, P. E., and Stork, D. G. (2006). *Pattern Classification*, volume 2. Wiley, 2. edition.

Floreano, D. and Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press.

Horner, A. and Goldberg, D. (1991). Genetic algorithms and computer-assisted music composition. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 437–441.

Hulse, S., Takeuchi, A., and Braaten, R. (1992). Perceptual invariances in the comparative psychology of music. *Music Perception: An Interdisciplinary Journal*, 10(2):151–184.

Jensen, J. H. (2010). Evolutionary Music Composition based on Zipf's Law. Technical report, Norwegian University of Science and Technology, NTNU.

Johanson, B. and Poli, R. (1998). GP-Music : An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters. In *Proceedings of the Third Annual Conference: Genetic Programming*.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

Machado, P. and Cardoso, A. (2002). All the truth about NEvAr. *Applied Intelligence*, 16(2):101–118.

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

Manaris, B., Roos, P., Machado, P., Krehbiel, D., Pellicoro, L., and Romero, J. (2007). A Corpus-Based Hybrid Approach to Music Analysis and Composition. In *Proceedings of the 22nd national conference on Artificial intelligence*. AAAI Press.

Manaris, B., Vaughan, D., Wagner, C., Romero, J., and Davis, R. B. (2003). Evolutionary Music and the Zipf-Mandelbrot Law: Developing Fitness Functions for Pleasant Music. In *Proceedings of EvoMUSART2003 - 1st European Workshop on Evolutionary Music and Art*, pages 522–534, Berlin. Springer-Verlag.

Minsky, M. L. and Laske, O. (1992). A conversation with Marvin Minsky. *AI Magazine*, 13(3):31–45.

Miranda, E. R. and Biles, J. A. (2007). *Evolutionary Computer Music*. Number 7. Springer.

Özcan, E. and Erçal, T. (2008). A Genetic Algorithm for Generating Improvised Music. *Lecture Notes in Computer Science*, 4926/2008:266 – 277.

Papadopoulos, G. and Wiggins, G. (1998). A genetic algorithm for the generation of jazz melodies. In *Proceedings of STeP*.

Phon-Amnuaisuk, S., Tuson, A., and Wiggins, G. (1999). Evolving musical harmonisation. In *International Conference on Adaptive and Natural Computing Algorithms*.

Ritossa, D. A. and Rickard, N. S. (2004). The Relative Utility of 'Pleasantness' and 'Liking' Dimensions in Predicting the Emotions Expressed by Music. *Psychology of Music*, 32(1):5–22.

Secretan, J., Beato, N., D Ambrosio, D., Rodriguez, A., Campbell, A., and Stanley, K. (2008). Picbreeder: evolving pictures collaboratively online. In *CHI 2008*, pages 1759–1768, Florence. ACM.

Sims, K. (1991). Artificial evolution for computer graphics. *Computer Graphics*, 25(4):319–328.

Vossa, R. F. and Clarke, J. (1978). "1/f noise" in music: Music from 1/f noise. *J. Acoust. Soc. Am*, 63(1):258.

Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *International Conference on Machine Learning (ICML)*.

Zipf, G. K. (1949). *Human Behaviour and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley.

Appendix

The evolutionary music system was built in the programming language *Python*[1], and a number of third-party libraries were employed:

- *numpy*[2] provided various computational tools.
- *matplotlib*[3] was used to create the various plots.
- *pythonmidi*[4] allowed parsing of MIDI-encoded music files.
- *celery*[5] was employed to run the experiments in parallel.

Experiments were run on the Kongull high performance Linux cluster at NTNU, which sports 98 nodes with 12 available processor cores each. By having 30 evolutionary runs in parallel, runtime was reduced from 6 hours to about 15 minutes.

Analysis of existing music was performed on high quality MIDI files donated by the Classical Archives (`www.classicalarchives.com`). The only exception is the song Let it Be, which was transcribed by the author since no high quality MIDI recordings could be found.

[1] Python Programming Language – `http://python.org/`
[2] Scientific Computing Tools For Python – `http://numpy.scipy.org/`
[3] matplotlib: python plotting – `http://matplotlib.sourceforge.net/`
[4] Python MIDI Package – `http://www.mxm.dk/products/public/pythonmidi`
[5] Celery Distributed Task Queue – `http://celeryq.org/`