$\mu Tunes$: Massively Multi-User Collaborative Composition of Music using Genetic Algorithms CM3202 One Semester Individual Project – 40 Credits

Robin Hawkins, 1125589

Supervisor: Dr. K. Sidorov
Moderator: Prof. Martin

May 6, 2014

Contents

List of Figures

List of Tables

Part I

Abstract

In this successful project, I have allowed hundreds of users to contribute to the composition of music by interactive evolution and analysed the hidden preferences which led to the final outcome. High levels of participation produced interesting and well-developed melodies and allowed for clear and convincing analysis of the music produced and the decisions which produced it.

Appreciation of music is a near-universal human characteristic. It is simultaneously a basic human activity and a supreme cultural achievement. Music composition, however, is a process typically restricted to a minority by two factors:

- it is extremely difficult to include the input of more than a handful of individuals
- these individuals each require an understanding of the fundamentals of music theory in order to contribute

Humans have an innate understanding of music which can be leveraged to surpass these barriers, using an interactive genetic algorithm. By ranking a population of musical ideas, many people's wills can sway the evolution of a collection of melodies. Using standard genetic crossover techniques, popular melodies and musical ideas can be perpetuated and propagated until we have a highly musical population, driven and shaped by many people regardless of their musical knowledge, ability and proximity to one another.

Using a web interface, participants compare pairs of melodies, submitting a decision as to which they prefer. These pair-wise comparisons will form the basis of a ranking system for a Genetic Algorithm. New melodies will be created based on the rankings, and over time the population will evolve to become more musical.

The evolution of the population will also provide us with insight into what humans perceive as musical. The user base will consist of musical and non-musical people alike, allowing us to analyse what is agreed upon as good music by all.

Part II

Acknowledgements

- Thanks to the Computer Music“Academy” of Cardiff University. Kirill Sidorov, Dave Marshall, Andrew Jones and Lawrence Semmens. It has been a privilege to have been involved!
- Thanks to Robert Evans for responding so quickly to my many and varied requests for bizarre server set-ups.
- Thanks to Ellen Keld, for trying to be interested, listening even when you are not, and for letting slide the fact that for the duration of this project, I have done almost nothing around the flat.
- Thanks to everyone to helped by participating in my project. Without you, this would be a much shorter report.

Part III

Body

Chapter 1

Introduction

1.1 Music - The Problem Domain

Many problems are known to have optimum solutions. The problem of creating great art is not one of them. This may be for lack of understanding of the problem, but for the purposes of this project, it is safe to assume that anything involving human taste is subjective; a matter of opinion. With respect to music, that means that there is no one perfect piece, and no one perfect genre. However, there are many properties of highly-acclaimed music on which human opinion agrees and many features converge to a large extent. Taking a broad view, people will generally agree that a piece sounds more musical when it conforms to a musical scale. Scrutinising a piece more closely, people's musical intuitions and biases predispose them to expect some notes more than others to follow particular musical patterns, forming what in Grachten, 2004 [8] are referred to as "structures".

1.2 Project Outline

The purpose of this project is to evolve music by a series of pair-wise rankings. By this means, I have built a system for collaborative composition of music requiring no musical ability to use, and to analyse human preferences and perception of musicality in an evolutionary context.

1.3 Interactive Genetic Algorithms

Genetic Algorithms comprise a category of heuristic search which mimic natural selection in order to approximate a solution to a problem. They require potential solutions to be represented in a way amenable to modification by operators such as crossover and mutation. Generally, crossover allows good ideas to propagate through a population, while mutation allows exploration of the space adjacent to a given solution. A fitness function assesses these solutions and assigns them a fitness value, by which they may be ranked. For well-defined problems, where solutions may be assessed objectively against clear criteria, this standard genetic algorithm as outlined above have proven to be very effective at finding local and global optima. However, the domain of music does not allow for clear, objective assessment of what is "good" and what is "bad", but rather depends on consensus of human opinion.

Interactive genetic algorithms (IGAs) lend themselves to capturing this consensus or group leaning, described by Yokohama [23] as *Kansei* (I will use this term in this sense throughout). IGAs replace an automated fitness function with one or more humans, truly allowing evolution to be driven by human opinion. This incorporates the human strength of being able to analyse music instinctively and in sophisticated ways, but brings several human weaknesses with it. The main obstacles here are threefold: time, consistency (noise) and fatigue.

Time

Humans assess information more slowly than computers. Generally, this is not a small difference, but one of orders of magnitude, making human interaction the biggest bottleneck in an IGA. This is a necessary trade-off in order to effectively capture human preferences, but it was a highly concerning risk factor which threatened to limit the amount of data I could collect.

Consistency

When working to well-defined rules, computers can be perfectly consistent. Humans offer no such guarantee, especially when, instead of assessing information against strict criteria, they are often led by sentiment and emotion. This is particularly true for music. For this reason, we are almost guaranteed to receive noisy data (in the form of contradictory results). This problem is compounded when more than one person is involved. Two completely valid opinions can be in direct opposition to one another. This necessitates finding a way of capturing the general *Kansei* of the users.

Consistency between users is well acknowledged to be unlikely, which is one of the primary motivations for this project. This did not concern me greatly as I was seeking a consensus.

Fatigue
People get tired, and only have a finite ability to assess or compare information, especially when they find it unrewarding. Fatigued people will not assess things properly and will be less inclined to keep assessing things, a fact which in turn will worsen the quality and quantity of the data we receive. Consideration of how to overcome this issue greatly influences the design of the user interface.

Effective implementation of a massively multi-user interactive genetic algorithm poses some non-trivial challenges. We have observed that many of these arise solely from the involvement of humans. In particular, fatigue. This is identified as a problem in IGAs generally, but many of those projects had an additional level of protection in that they had a captive user base. This project operates with no such safety net, relying on a voluntary user base to achieve its goal. The algorithm I designed was heavily shaped by the way in which I chose to mitigate this fatigue. Additionally, we cannot rule out noise when relying on humans for assessment, so I had to construct a system which could withstand some noise.

1.4 Pair-wise ranking

The minimum I felt that I could ask a user to do in order to contribute usefully to this project was to compare two melodies and choose which they prefer. For this reason, I chose this as the means by which users can interact with and influence evolution. I believe it strikes the best balance between fatigue-mitigation and gaining useful information.

I could have asked for less. I could have asked users simply to state whether or not they liked a melody. However, assessing melodies in isolation like this is unproductive, as when ranking we are interested in their relative fitness. This method also helps to minimise noise when compared to an objective assessment of each melody. For example, if we asked a person to rate a melody out of ten for musicality, although they may only be presented with one melody at a time, we are tacitly asking them to compare this melody with every other melody from our population that they've heard (if they are not doing so, then their assessment is worthless as a means of ranking). As this would involve another comparison for each additional melody they ranked, I would effectively be raising the bar to participation as it happened. Similarly, if we ask them to rank more than two melodies, we are expressly asking them to remember more melodies. So we see that pair-wise ranking offers a guaranteed maximum amount to consider, thus minimising effort — a participant can ignore everything they heard before the two melodies they are currently considering. Similarly, it is more likely that two people will prefer melody A over melody B that both award melody A an identical score in all cases.

So, the results of these pair-wise comparisons are the atomic units of data which will drive my project forward. They ask little of the user while still yielding useful information which is likely to be consistent at the level of an individual user. Noise from inconsistencies (i.e., conflicting opinions) at a multi-user level can be diminished or mitigated entirely by seeking consensus in the data and ranking based on that. In comparison to methods of querying users employed by other projects, such as DarwinTunes [15], which asks users for five categories of information, this method is extremely straighforward. I gathered information regarding what different aspects of the melodies were preferred from an analysis of the evolution.

This technique of pairwise comparison has been used before with success in Yokohama/Takenouchi 2012 [23] to extract the user group's *Kansei*, indicating that it is a robust enough method for the *μTunes* project.

1.5 Particiption in the project: who and how

As part of the remit of this project is to determine generally what people find musical, it is beneficial to have a user base from as broad a demographic as possible. Cultural influences as well as general level of musical education colour how people perceive music and what they perceive as musical, so ideally I want a user base broad enough that these factors are secondary to more generic trends. The global, ubiquitous and largely uncensored nature of the internet makes it an ideal medium through which to involve people. Computers do not sleep and the project is autonomous, so my time is not a restricting factor.

A true test of a massively collaborative compositional tool was to have people who had never met give their input and arrive at a musical outcome without any other collaboration or even communication. By making the project available online that test can be undertaken. Thus, my user base is limited to everyone who can access the internet and play sounds.

This may, to some extent, rule out the elderly as a group. They are less likely to be internet users. This is disappointing as they have been conditioned by different kinds of music to younger generations, and their idea of what is music may not be represented by the end results of the project.

1.6 Goals and expectations

This project aims to do several things:

Provide a means of massively collaborative composition of music for which level of musical ability is no bar to entry This project has the scope to prove that this is possible, at least in principal. The ways in which I was forced by time constraints to limit it means that the results, rather than being a complete composition, will be a set of melodies from which to construct a larger piece.

Demonstrate the Extent to which there is Consensus on what Musicality is The prevalence of Western music throughout the world colours people's perceptions of what music is. Already this narrows down what I would expect to find. My hypothesis is that the population will converge on diatonic melodies (those melodies which conform to Western major or minor scales). This prediction is relatively easy to measure, as it is just the extent to which a melody belongs to a certain set of notes. Similarly, almost all music makes heavy use of repetition, which can manifest itself in many forms, such as rhythm or patterns of differences between notes. It would be unsurprising to find that the population has converged on melodies with some repetitious qualities. Beyond these two features of music, there are many more subtle

indicators for which I will test.
Test the Robustness of Noisy Pair-wise Comparisons as a Means of Data Gathering Given that we are looking for preferences that are generally true, I would expect this method to strike a good balance between ease of user participation and effective data collection.

1.7 Limiting factors, potential pitfalls, and risks

The greatest risks to the project lay in the fact that user participation is voluntary and remote. I cannot predict in advance how many people will take part, encourage their level of participation, or direct *how* they participate. Possible scenarios related to the human aspect that could derail the project are:

1. Insufficient participation to drive the project
2. Insufficient input from each user
3. Deliberate noise from users (i.e., trolling)

If the massively multi-user aspect of the project were not realised, it would be very difficult to gain anything other than methodological data from the project. It may be possible to prove that it works in principle by driving it myself, but I would certainly not be able to make any broader judgements about what appeals to people as music and what does not.

The massively multi-user aspect of the project also gives rise to technical difficulties. It is difficult to predict what bugs or algorithmic flaws may manifest themselves under the weight of heavy traffic. By considering the possibilities, I tried to design a system which could cope with heavy loads and unpredictable behaviour. Subtle problems had the potential to skew the data. For example, if a user opened a page to compare a pair of melodies, but did not submit their decision until after a melody had been removed from the system, how would this affect the system? What would happen if they did not submit a decision at all?

These unknowns necessitated a certain amount of probabilistic behaviour from the system in order to cope. Balancing reliability and robustness against accuracy and utility was one of the greater challenges of the project.

Bearing in mind these considerations, I foresaw the following difficulties:

1. Driving participation in the project
2. Encouraging and rewarding user participation in the project
3. Extracting meaningful data from potentially unreliable, inconsistent user input
4. Ranking melodies with incomplete data

Time constraints are a factor which affected many decisions made during this project. The density of user responses (i.e., the enthusiasm of participants) was so important because the time I had to collect data was finite and fixed. This led me to narrow the scope of the music that my project was capable of producing in several ways.

Chapter 2

Background

Computer-aided generation of music is not a new idea, and nor is using a Genetic Algorithm to accomplish this task.

Previous projects fall into two broad categories: those that use the rules of music and quantifiable data (e.g., Evolutionary Music Composer by Khalifa and GP-Music by Johanson [12, 10]) to rate fitness and direct evolution, and those which use human feedback to hone compositions towards musicality (e.g., Evolutionary Interactive Music Composition by Wu et al ,[24], DarwinTunes by MacCallum et al [15]). *μTunes* joins the latter category. Wu et al [24] uses small musical ideas as the individuals in its population and builds melodies from them, and DarwinTunes [15] deals with audio samples, combining them to produce more pleasing one.

The nearest thing to this project is DarwinTunes [15]. However, that project seems to be focussed on timbre and textural features of music rather than melody. *μTunes*' focus on melody distinguishes it from DarwinTunes [15] in both the way it operates and the data we are gathering. *μTunes* distinguishes itself by occupying middle ground - exposing participants directly to the individuals we are manipulating, unlike Wu et al [24], but still treating the melodic passages as musical instructions rather than rendered audio.

Both *μTunes* and DarwinTunes rely on public participation, but DarwinTunes focuses more on timbre. I am trying to negate the effects of timbre entirely in order to focus solely on defined sequences of notes. Also, the manner of participation in DarwinTunes is by rating the aesthetic qualities of a section of audio, whereas mine is by comparison. Most importantly, the creators of DarwinTunes are approaching it from a biological background, with a particular interest in how memes propagate (or rather, how humans propagate memes). I am approching this project as a novel means of data-gathering for analyses of features of melody.

Pairwise comparison has been used by Wu et al [24] in Interactive Genetic Algorithms as a “consensus building method that determines the win-lose results for the tournament using the votes of multiple people”. Knowing that it is capable of yielding data which pleases a majority of people indicates that it is an appropriate method for data collection in the *µTunes* project.

Although the main elements of my project (a large, decentralised user base, evolution of melodic pattern, pairwise comparison, use of social media) are not novel, the combination appears to be.

Chapter 3

Models, Methods and Algorithms

An interactive genetic algorithm is a generic framework which requires bespoke components to tailor it to a specific function. These are:

- A solution representation
- A means of generating an initial population
- A means for participants to rank/assess the fitness of the constituent solutions of the population
- A condition to initiate the creation of a new generation
- A means of creating a new generation (including the tools to do so)
- A means of assessing when the process is finished

These are discussed in detail below.

3.1 The Genetic Algorithm

Here is a brief summary of how the algorithm works, followed by an in-depth look at the components of the algorithm.

3.1.1 Walk-through

This is a conceptual model of my algorithm.

1. Two melodies are selected for comparison. The first is selected in order, the second at random from the remaining population

2. The two melodies, A and B are presented to the user for comparison. They listen, and select the one they prefer. For the sake of example, we will say they preferred A.

3. A's score is incremented by one, B's score is decremented by one, to reflect which was preferred.

4. The population is sorted using the scores of the individual melodies.

5. With some probability, the next generation is generated. Offspring of the top $n\%$ by rank are created by standard genetic operators of crossover and mutation.

6. The new offspring replace the bottom $n\%$ by rank.

7. The process repeats.

3.1.2 Representation

Genotype

As discussed in Li et al, Alfonseca et al [13, 17], the key of a melody does not affect its recognisability. Likewise, although it is argued that each key signature has its own character (e.g., melancholic, jubilant, such as the characterisations of key by Berlioz in his Grand trait d'instrumentation et d'orchestration modernes, Op.10 (1844) [6], examples in Figure 3.1), key does not uniquely define a melody. As such, in terms of pitch, a melody can be considered invariant with respect to key and defined only by the *relative* pitches between notes. For this reason, I have elected to represent note sequences as a series of offsets. The pitch of the *nth* note is an offset from the pitch of the first note, given by:

$$p(n) = C + \sum_{i=1}^{n} O(i) \tag{3.1}$$

where O is an ordered list of offsets, $O(i)$ is the i^{th} offset in semitones, and C is an arbitrary constant dictating the absolute pitch of the first note. In the representation, $O(1)$ is always zero, so that C may dictate the key, or "seed" the list of notes.

The rhythm of the melody is dictated by assigning a duration to each offset. Each duration is an integer value, representing a scalar by which some atomic unit of duration should be multiplied.

The following sequence represents "Mary Had a Little Lamb", with a semiquaver (16th note) as the atomic unit of duration:

> 0:4,-2:4,-2:4,2:4,2:4,0:4,0:8,-2:4,0:4,0:8,2:4,-2:4,-2:8,4:4,-2:4,-2:4,2:4,2:4,0:4,0:8,-2:4,0:4,2:4,-2:4,-2:8,

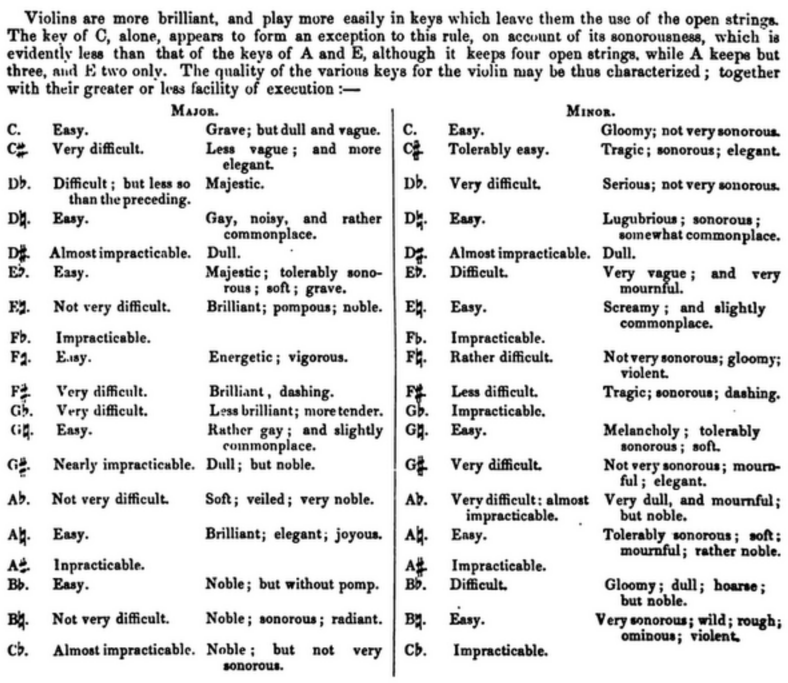
Variation on this, such as the **mint representation [1], are fairly common. It has an intuitive appeal to a musician; I find that I can site-read it to play these melodies on various instruments. Being human-readable is an advantage in a genotype representation as it is easier to confirm that it is being manipulated as it should be.

Phenotype

The melodies are presented to the user as audio files dynamically generated by MIDI synthesiser. The melodies are played on a piano, with the starting note set to middle C. I chose to avoid an instrument with strong cultural associations or connotation of genre, such as a banjo or a sitar, as I wanted to avoid steering the listener towards an outcome, either consciously or subconsciously. The piano, being used in a wide variety of genres of music, is less likely to colour a listener's perception of a melody. I chose to start on middle C for the same reason; that it is in the middle of the typical range of music and should also help to avoid colouring a listener's perception.

Other restrictions

Melodies can be dictated, affected, coloured and nuanced by any and all of the following factors: relative pitch, rhythm, repetition, key, timbre, speed, dynamics, swing, stress and harmony. I have identified the first two as the most significant with regard to quality and identifiability of a melody. The



Violins are more brilliant, and play more easily in keys which leave them the use of the open strings. The key of C, alone, appears to form an exception to this rule, on account of its sonorousness, which is evidently less than that of the keys of A and E, although it keeps four open strings, while A keeps but three, and E two only. The quality of the various keys for the violin may be thus characterized; together with their greater or less facility of execution :—

Major.			Minor.		
C.	Easy.	Grave; but dull and vague.	C.	Easy.	Gloomy; not very sonorous.
C♯.	Very difficult.	Less vague; and more elegant.	C♯.	Tolerably easy.	Tragic; sonorous; elegant.
D♭.	Difficult; but less so than the preceding.	Majestic.	D♭.	Very difficult.	Serious; not very sonorous.
D♮.	Easy.	Gay, noisy, and rather commonplace.	D♮.	Easy.	Lugubrious; sonorous; somewhat commonplace.
D♯.	Almost impracticable.	Dull.	D♯.	Almost impracticable.	Dull.
E♭.	Easy.	Majestic; tolerably sonorous; soft; grave.	E♭.	Difficult.	Very vague; and very mournful.
E♮.	Not very difficult.	Brilliant; pompous; noble.	E♮.	Easy.	Screamy; and slightly commonplace.
F♭.	Impracticable.		F♭.	Impracticable.	
F♮.	Easy.	Energetic; vigorous.	F♮.	Rather difficult.	Not very sonorous; gloomy; violent.
F♯.	Very difficult.	Brilliant, dashing.	F♯.	Less difficult.	Tragic; sonorous; dashing.
G♭.	Very difficult.	Less brilliant; more tender.	G♭.	Impracticable.	
G♮.	Easy.	Rather gay; and slightly commonplace.	G♮.	Easy.	Melancholy; tolerably sonorous; soft.
G♯.	Nearly impracticable.	Dull; but noble.	G♯.	Very difficult.	Not very sonorous; mournful; elegant.
A♭.	Not very difficult.	Soft; veiled; very noble.	A♭.	Very difficult: almost impracticable.	Very dull, and mournful; but noble.
A♮.	Easy.	Brilliant; elegant; joyous.	A♮.	Easy.	Tolerably sonorous; soft; mournful; rather noble.
A♯.	Inpracticable.		A♯.	Impracticable.	
B♭.	Easy.	Noble; but without pomp.	B♭.	Difficult.	Gloomy; dull; hoarse; but noble.
B♮.	Not very difficult.	Noble; sonorous; radiant.	B♮.	Easy.	Very sonorous; wild; rough; ominous; violent.
C♭.	Almost impracticable.	Noble; but not very sonorous.	C♭.	Impracticable.	

Figure 3.1: Characters of Different Key Signatures when Played by Violins

consideration of melodies is limited to these factors within the scope of this project. This is because all the other factors can change and a melody can still be recognisable. It is also for simplicity's sake.

3.1.3 Generation

Each melody within the initial population is generated randomly, with certain restrictions and biases. Each initial melody is four bars of four crotchets. This is a small amount of music, but sufficient to express a strong motif. The repetitive nature of much music means that a concise passage can contribute multiple times, directly and in variation, to a piece. For these reasons, the length of a phrase is not proportional to its musicality or level of contribution to a larger piece. Additionally, the brevity of these passage also helps mitigate user fatigue, which, as mentioned previously, is a huge bottleneck and very real problem in interactive genetic algorithms. Longer melodies would have been more difficult for a user to compare, and would not have offered a tangible gain in terms of the information offered. Rather the contrary: a longer passage would have contained more information, and so it would be harder to determine what in particular a user is selecting for or against.

The offsets, which are integers, are calculated as follows: a discrete Gaussian distribution is generated to an arbitrary distance and variance from the mean. These values translate to the magnitude of the maximum desired offset (the range) and the disjunctness of the melody overall (the likelihood of a large or small jump between notes; the gradient of the Gaussian curve). Note: I only generated half a distribution. As I only needed to calculate the magnitude of the offset. The direction of the offset is decided later. The distribution D is calculated thusly:

$$D(x) = e^{-x^2/2\sigma^2} \tag{3.2}$$

Then the normalised distribution $D_{normalised}$ is calculated.

$$\sum_{x=0}^{\frac{\text{range}}{2}} D(x) = 1 \tag{3.3}$$

$$D_{\text{normalised}]}(x) = \frac{D(x)}{\sum\limits_{d \in D} d} \tag{3.4}$$

For each offset in the melody, a random number $n \in [0, 1]$ is generated. The offset is given as the minimum number of items from $D_{\text{normalised}}$ such that their sum exceeds n, minus one. Or:

$$\text{Offset} = \min_{\left\{y : n < \sum_{x=0}^{y} D(x)\right\}} \tag{3.5}$$

So, for a range of 5 and some arbitrary sigma:

Generate and normalise distribution

$$0.7, 0.2, 0.1 \tag{3.6}$$

Generate random number N between 0 and 1

0.836275462. . . etc.

Work through distribution until cumulative sum exceeds N

- Position 0. Cumulative sum = 0.7 Greater than N? No.
- Position 1. Cumulative sum = 0.9 Greater than N? Yes.
- Return 1.

Our offset from the previous note is 1.

The direction of the offset is decided at random, with equal probability given to up or down. This is acheived by multiplying the offset by -1 with $P(\text{offset} = \text{offset} \times -1) = 0.5$ so melodies can occasionally descend!

There are two constraints placed on these melodies as they are generated. The use of a Gaussian distribution to generate offsets is one. The other is a check to make sure that the total melody remains within an octave of where it starts. This is done by keeping a cumulative sum of the offsets. Should an offset take the melody further than an 8tve either side of the first note, it will be discarded and a new one generated.

I put these checks in place after trying generating melodies completely at random. Melodies generated in this manner were so poor that I felt it necessary to increase the likelihood of generating something in some way musically appealing. This was probably useful in decreasing user fatigue, as more melodies had more redeeming features.

The range was set to 12, ie a maximum jump of an 8tve (twelve semitones, a doubling or halving of the frequency) up or down, and sigma to 7. I used a Gaussian distribution rather than the actual probability distributions for a given form of music as I wanted to avoid stylistic bias.

3.1.4 Operators

Genetic operators are blind. They don't know whether or not their effects will improve or damage a given solution. This is their strength and weakness, as they produce solutions a conscious being with intuition into a problem area may not; these solutions may be surprisingly good or predictably bad. However, it is not the responsibility of these operators to improve the general quality of a population (this role falls to the selection operator, I would argue), but to provide options, avenues and opportunities for a population's continued growth and evolution. The responsibilities of these operators should be to forge new information from old, without being too destructive or restrictive, and to avoid bias. The latter point is particularly significant within the domain of music and this project. The subjective quality of music is strongly tied to genre and the traits and motifs of genre; that is to say, people who prefer country music over rock will probably choose average country music over good rock [1]. I don't expect musical biases to manifest themselves in as sophisticated a fashion as obviously belonging to a particular genre, but I want to avoid restricting the direction of evolution as much as possible by my choice of operator. That role falls to the human participants.

Crossover

I chose to use one-point crossover with a uniform distribution over the atomic duration (as mentioned earlier, this is the chosen shortest note-length for the project). The effect of this is that melodies are not merely split between notes (by which I mean my representation's offset:duration pairs), but can be split in the middle of notes, down to a granularity of the atomic duration.

For example:

0:4,2:4,1:4,-2:4,0:4

has a length of 20 atomic note-lengths (5 notes of duration 4). This gives us 19 points at which to split the melody. For example, let's say that we split this melody for crossover at point 11. This is what it would look like:

First half: 0:4,2:4,1:3

Second half: ?:1,-2:4,0:4

In this case, the question arises: does the second half of the split note retain the offset of the original, or simply become a repeat of whatever ends up preceding it (offset of zero)? Neither outcome is more intuitive, nor clearly better than the other; in some circumstances a repeat is beneficial, in others, not. In many cases, opinions on this will probably be completely subjective,

[1]Incorrectly, I might add.

or it will be context-dependent. Also, as my representation deals with offsets rather than fixed pitches, each note affects those succeeding it, altering their absolute pitch in the audible melody (i.e., the phenotype). Again, we can't tell in advance if a melody will be better one way or another. As such, I decided that either event occurring with $P(\text{event}) = 0.5$ was the fairest way to proceed, as it ruled out nothing.

From this choice of crossover method, a problem arises. The later the generation, the higher the chance of a given melody being comprised of short notes. Stylistic bias in a population should be driven solely by the human participants in the project, not the recombination operators! To rectify this situation, I introduced a step where, with $P(\text{event}) = 0.5$ I merged repeats into the note they are repeating. In this case

0:4,1:4,0:4

would become

0:4,1:8

This allows average note duration to lengthen as well as shorten, thus counteracting the bias from the previous step. However, I recognise that a further subtle bias has now been introduced: pitch information is being retained at the expense of rhythmic information. While this is a bias, I consider pitch information to be more significant to the unique identifiability of a melody. Also, the effect is not a large one, so I deemed it to be of negligible risk to the outcome of the project.

The positions before the first note and after the last note are ignored as points for crossover for the obvious reason that they would simply reproduce the parent melodies. However, there is often another point which will also often have this effect. Because the offset of the first note in each melody is always zero, if two melodies have a first note of the same duration, crossover between the first and second notes will reproduce the parents exactly, even though genetic material from different sources has actually been combined. It seemed to me that this problem is insignificant for two reasons:

1. it is statistically fairly unlikely

2. propagation of the genetic material of successful melodies is no bad thing, even if no new information is gained.

This circumstance will probably not retard the process of evolution in a noticeable way due to its infrequency.

Mutation

In Genetic Algorithms, mutation allows an exploration of the space around a given solution. Special consideration needs to given to the fact that in my representation, any change to a single gene affects the expression of those following it. As my crossover operator necessitated some extra procedures that affected rhythm, I have limited mutation to offsets. As offspring are produced in pairs, I chose one offspring from each pair at random to undergo mutation. A single mutation pitch-shifts all of the melody that follows it. It is impossible to know whether this will improve the melody or not, but my intuition as a musician warns me away from very large pitch shifts, as they are likely to sound jarring and unmusical. As such, mutation either increases or decreases an offset by 1, with equal probability.

I could have made mutation a much more subtle effect by counteracting the pitch-shift. This would have manifested itself as a mutation followed by a mutation of the next note of the same magnitude in the opposite direction. As with much in this project, there is no rule as to which way is better; in some cases one way will result in a net increase in musicality, in others it won't. However, I postulated that worse melodies have more to gain from broad changes than subtle ones. As the initial population had low musicality, I opted for the method of mutation with more far-reaching effects.

3.1.5 Ranking

The need to mitigate user fatigue led to my decision to rank melodies through a series of pairwise comparisons. This a non-trivial problem, compounded by the fact that some noise (i.e., contradictory votes) is almost guaranteed and that it is hard to ensure that enough data is gathered for an accurate ranking. I have no guarantees as to how many comparisons a user will make, and whether or not a user will even send their vote having listened to a pair of melodies. Another important consideration was that I did not know in advance how many participants I would have and how active they would be. Therefore speed of ranking was a concern which drove some of these ideas. I considered several approaches, three of which I will outline now. The one which I eventually selected was the least exotic of the possibilities.

Sort-style ranking

Here I considered seeding the population with an arbitrary ranking, with lower ranks (1,2,etc.) corresponding to better melodies. Upon being presented with a pair of melodies, a user would select the one they think is better. If the more preferred melody had a higher rank than the less preferred melody, they would switch ranks. Eventually, the ranking would broadly reflect a consensus of user opinions.

Probably the greatest problem with this method is its massive vulnerability to noise. Say we have a population of 10 melodies. Let's say that every user so far has rated the 1-ranked melody as the best. Then the last user to engage in the project compares the 1-ranked melody with the 8-ranked and prefers the 8-ranked. This outlying opinion supersedes all previous opinions and misrepresents the group opinion.

In tests, this method did achieve convergence, but slowly. In an environment where more consistency could be achieved, it could perform quite well - for example, in a system with only one user.

Points Threshold

This technique relies on the fact that we only need rankings to be roughly correct to generate the next generation. In this project, where the offspring of the top x% replaces the bottom x%, as long as melodies in these bounds are correctly ranked the process of evolution will occur. When two melodies are compared, the "winner" (preferred melody) is allocated one point, and the loser has a point taken away. Melodies are ranked by sorting according to accumulated points, with ties being broken randomly. Although I came up with this method myself, it has been explored before (Balanced Rank Estimation, Wauthier [22]). I realised that the most liked and most disliked individuals in a population would quickly become apparent. Once the difference between the highest and lowest ranking melodies reached a certain point, the creation of the next generation would be triggered. This method was attractive to me as it was fairly resistant to noise and the threshold could be lowered to speed up the process of evolution or raised to increase the accuracy of the rankings. The threshold should be set so as to strike a balance between the two. It struck me as an appropriate method for group composition, as reaching the threshold indicates a certain level of consensus, based on which a composition can be advanced.

Indeed, it performed quickly (almost five times as quickly) in tests than sort-style ranking.

Balanced Rank Estimation with Stochastically Triggered Creation of New Generations

This is the method I finally elected to use. Due to its simplicity and robustness I used Balanced Rank Estimation to rank the melodies, but instead of waiting for the score spread to reach a certain threshold, the next generation is created with probability

$$P(\text{nextgeneration}) = \frac{1}{\gamma n \log \gamma n} \tag{3.7}$$

after each comparison, where n is the population size and γ is a scaling factor used to speed up or slow down evolution.

Balanced rank estimation populates the extremes of the rankings first, which is useful to us. Poorly thought-of melodies will soon end up with a negative score and be pushed towards the extinction zone at the bottom of the rankings. Melodies which have not been compared, or only compared once or twice will remain in the middle of the rankings, to survive into the next generation. This is positive as it avoids throwing away good ideas by accident. Retaining bad ideas is not problematic as they will eventually be ranked enough to be pushed to the bottom of the table and killed off.

It is significant that scores are retained from one generation to the next. I chose to do this partly because of the stochastic triggering of the generation of new melodies; in theory there could be a single comparison to rank an entire generation (although this is quite unlikely). It does, however, create a legacy effect where previously successful melodies are higher in the rankings.

3.1.6 Generating the next generation

With the probability described above, after a comparison is submitted, a new generation will be created. This is achieved by taking the top 20% of the population by rank, performing crossover and other procedures as described above on random pairs of melodies from this selection such that each melody is crossed over only once. The children of these melodies (i.e., those individuals to be introduced in the new generation) replace the bottom 20% by rank, being inserted into the positions previously occupied by the bottom 20%. Position in this context is used to determine which melodies are presented for comparison next, and should not be confused with rank.

3.1.7 Stopping Condition

A strict stopping condition has not been defined for the purposes of this project. Part of the difficulty in doing so for this application of Interactive Genetic Algorithms is that convergence on just a single idea may rule out other good material. Conversely, we cannot assume that $\mu Tunes$ will always produce lots of good ideas. Music being a subjective experience, I consider it wiser to allow the system controller to monitor the level of convergence (by analysis such as I have undertaken) and when entropy begins to decrease, either cherry-pick material they like from the emerging ideas or continue to run the experiment. As such, the stopping condition is user-defined. If it runs to a point where diversity amongst melodies has become virtually non-existent, its utility will have been exhausted and it can be said to have run its course.

3.1.8 Summary of parameters

Population Size Number of individuals in the population. Set to 100, in an attempt to balance variety and speed of convergence.

Gamma (γ) Scaling factor affecting the frequency with which the next generation is created. At $\gamma = 1$ then there should be enough comparisons to accurately sort the population. Reducing gamma represents a trade-off between accuracy of ranking and speed of evolution. Gamma was set to 0.5, to expedite evolution.

3.1.9 Limitations of the algorithm

Setting up this project as a web application was the obvious choice, given that ubiquitous, cheap and well-established web technology was well suited to transfer of media and information. Use of the web presents certain problems, though. Given the remote nature of the web, I could not monitor users behaviour. Thus, I could not ensure that they will submit a comparison after two melodies are presented to them, and this makes it difficult to ensure that all melodies have equal or nearly equal opportunities to be ranked. Simply waiting for a preference to be submitted before selecting the next melody(s) to be compared is no better because, as a web-based system, it could receive a new request for melodies before a user has finished comparing the previously issued ones. For this reason, the choice of which melodies to present for comparison is partly stochastic. The semi-random nature of selection helps fill in the gaps left by unreturned responses, but does not guarantee fair selection.

The capacity for the emergence of repeated passages (phrases longer than two or three notes) in the population is inhibited by the crossover operators. These are deliberately naïve, picking a single point and then swapping first half for first half, second half for second half. A passage appearing in the first half of a parent melody will never appear in the second half of a child melody, and so for repetition of a passage to occur, it will have to emerge separately, in a different part of a different parent, and then end up being selected and crossed over with the right melody at the right point, which seems unlikely.

The emergence of rhythm is as a side-effect of operations to create children of a pair of melodies. The naïvety of this process leaves something to be desired and seems unsophisticated, in retrospect.

Chapter 4

Implementation

Many of the decisions in this project were made to try to limit the bottleneck created by involving human participants, and limit the extent to which fatigue effects their ability to participate. I knew from previous studies by Babbar et al, Llora et al, Manaris et al, and Takagi et al [5, 14, 16, 19, 20]) that the detrimental effects of fatigue are not to be taken lightly. For this reason it was important to me to make the experience for the user as pleasant, easy and unhindered as possible.

The easiest, most accessible way to make this project available to potential participants was as a web application. It also expanded the potential user-base to an international level and allowed people to participate via any internet-enabled device. The portion of the site accessible to the user consisted of three pages: the homepage, which contained a brief explanation of the project and the participant's potential roll in it, also detailing the prizes I purchased as an incentive to participate. The "compare" page, which allowed the user to listen to two melodies, select which they preferred, then submit their choice. On clicking submit, they would be given another two melodies to compare.

4.1 Technologies used

I had planned to use a Common Lisp based web framework called Weblocks [4]. It seemed to promise tight integration of front end, back end, data structures and logic, all manipulable programmatically. It may well be a good solution, but I was not able to find out; I found it so poorly documented that I made almost no progress with it and was forced to abandon it[1] in favour of a more familiar LAMP (Linux, Apache, MySQL, PHP) stack. The "P" in my stack stood solely for PHP, not Perl or Python. I also made some use of JavaScript for input checking and altering playback options.

My experiments with a lisp-based system had cost me quite a lot of time, for which reason I kept the structure and implementation of the site relatively simple. Beyond considerations of user-friendliness, the aesthetics of the site were unimportant to the project.

The site was developed on my own set-up, using a Raspberry Pi as a server. I did this as I did not have administrative privileges on the school's server, which was to host the site ultimately, and I wanted to know exactly what I needed to have installed on there before I asked, to make the process easier. Having two parallel also allowed me to test changes without affecting the running system and run other experiments (like my primary control experiment).

For analysis, MATLAB was chosen for friendly syntax and ability to process large arrays of data in useful ways quickly. It's ability to produce figures easily was also a boon.

For backing up and developing the code and this report, I used git version control and Dropbox, respectively.

4.2 Front-end and user experience

The site is still running at http://ontario.cs.cf.ac.uk/mutunes/ I do not consider myself web designer. For this reason I limited my aesthetic goals for the look and feel of the website to being unobtrusive, easy to use, and not ugly. I am reasonably happy that I achieved these goals, and heard no aesthetic complaints. Each of the three pages of the site share a heading and a menu. As is evident from figure 4.2, the design of the site is clean and spartan. I

[1]Lisp's lack of syntax is often touted as one of its great strengths, and indeed, it can be a very transparent language in which to implement an algorithm. However, in this situation, it has been used to develop a set of tools. Some syntax may have given my clues as to the usage and structure of the framework which were devoid in this case. Of course, proper documentation would have been equally acceptable.

did not want to draw too much focus from what I wanted the user to be doing.

Of more pertinence to the project are several conveniences for the user which attempt to mitigate fatigue. Listening to four-bar phrases repeatedly takes time, and can become tiresome. Also, there is no timbral variation between melodies as they are all played using a piano synthesiser. I introduced an option to alter the speed of playback, which was remembered for the duration of a session. There is an option for a user to declare themselves to be a musician or not. This selection was recorded, as it is of interest whether individuals with musical training made significantly different choices to non-musicians, and how they differed. This option was remembered by a cookie.

The "leave details" page is a simple two-field form preceded by an explanation of why to leave details (or rather, what incentives there are for the user). Again, the minimal nature of this form is a deliberate attempt to lower the bar to participation wherever possible, in keeping with the remit of this project to facilitate involvement with the minimum of ability. Leaving details was is not mandatory, but cannot be done unless the user has contributed to the project with at least one comparison.

4.2.1 Browser compatibility

Modern, up-to-date browsers make playing audio relatively simple, but not everyone is a modern, up-to-date user. As such, old versions of browsers which don't support newer features still abound. A perennial problem encountered by web developers everywhere is that Microsoft Internet Explorer insists on interpreting both HTML and Javascript emphslightly differently from the other major browsers, Chrome and Firefox. Part of my decision to keep the front-end design simple was driven by these problems - functionality is more important to my project than aesthetics, so I dedicated more of my time to working on it.

Audio in particular is the area of multimedia I wanted to deliver. According to the w3schools website [3], the ⟨ audio ⟩ tag is supported by all the major browsers, but support for formats varies. As all of the browsers support either WAVE or MP3, I elected to serve both and thereby support all modern browsers. As the audio files will be short, serving the bulkier WAVE format should not dramatically increase server load, but there is the lighter mp3 option for when it is possible to use it. This allows for "graceful failure" and covers the vast majority of modern browsers.

Given the vast amount of web browsing now done through tablets and mobile phones, designing a system which worked well on these devices was as important as designing for use on a traditional desktop or laptop computer. The main considerations were lower processing power and smaller screen size, but neither presented a particularly big challenge. Even lower-powered devices are designed to support multimedia. As I had kept my interface clean and simple, and very little processing is done in the browser, all I needed to do to adapt the experience for mobile was add a style sheet to compensate for different device orientations.

4.3 Back-end

My project's site and database were hosted by Cardiff University. The site and was primarily PHP-driven, with some javascript on the front end to enhance the user experience and perform input checks such as validating email addresses. The algorithm was enacted entirely in PHP with the state being stored by a MySQL database. I elected not to use a framework for several reasons, not least of which being that I was not already familiar with one. I had already lost time and had a frustrating experience using a lisp-based framework. I was wary of potentially using up a lot more time familiarising myself with another framework without being able to assess its appropriacy for the task in hand. The nature of the site required for this project was not particularly complex on the front end, where as the back end was relatively esoteric and specialised. As such, it seemed logical to write a lot of it by hand.

For the layout of the site, I wrote generic menus and headings and used PHP to dynamically construct the pages. I used a pre-existing PHP library [2] to generate midi files, but apart from this all the code uses either default PHP5 libraries or is my own work.

For control of the project, I wrote myself a password-protected content management page from which I could change the algorithm's parameters (such as γ, population size). This page also allowed me to view the rankings of the melodies and the total number of comparisons. There is also an "export" page which allows me to download the current state of the "melodies" and "battles" database tables in csv format.

4.3.1 Admin Pages

There are two password-protected pages to the site.

cms.php

This allows parameters to be changed, including the population size, γ (which affects the number of comparisons in between generations) and the percentage of the population to be replaced each generation. It also displays a table showing the current ranking of the melodies and other data about them. It also allows you to reset the system completely, generating new melodies. If you want to visit the page, email me and I will give them to you. If you should choose to do this, please don't modify any of the system or reset it.

export.php

This allows me to retrieve the contents of the "melodies" database and the "battles" database in csv format. Scripts here write this databases to a pair of CSV files, then render them available for download.

listen.php

This is not password-protected, nor strictly an admin page. It allows you to listen to any melody you want by selecting it's ID from the dropdown box and clicking"submit". I only created this page recently to make analysis of the melodies easier.

4.3.2 Database

I used a MySQL database to hold all information about the project including it's current state, all of the melodies which have been removed from the population, the results of individual comparisons and general project parameters. I tried to record the data in a way that events could be played back sequentially and evolution scrutinised closely. For this reason a record of all comparisons is kept.

The database contains the following four tables:

Table: generalProperties

Parameters for the project are stored here. I did this to centralise them, avoid hard-coding the parameters into PHP, and allow them to be more easily manipulated. This table has 9 fields, many of which are redundant in the final incarnation of my project:

`name` Just a field to use as a unique identifier for this set of properties.

`popsize` The size of the population.

totalComparisons A record of the total number of comparison results received. Now redundant, as it is equal to the number of rows in the 'battles' table.

`generation` The generation of the current population. Now redundant, as it is equal to the maximum value of the 'introducedAtPopulation' field in the 'melodies' table.

`thresholdForStopping` Originally included when I planned to calculate a termination condition for the project. Now redundant.

`threshold` Originally included when testing a previous incarnation of this project,using the 'Points Threshold' system as mentioned earlier. Now redundant.

`currentPosition` Used to determine one of a pair of melodies to be presented to the user. This attributed is incremented by 1 each time it is retrieved. One of the next pair of melodies to be presented to the user is given as

$$\text{Positio}n = \text{generalProperties.currentPosition mod generalProperties.popsize} \tag{4.1}$$

`percentToReplace` The percentage of the population to replace in a new generation. Also the percentage to become parents to the next generation.

`gamma` Scaling factor to change the rate of evolution. A value of 1 should allow for enough data for accurate ranking.

This table only has one row, corresponding to one project. It would likely be possible, with little adjustment, adapt this database to support multiple $\mu Tunes$projects simultaneously.

Table: battles

A record of comparisons and metadata about those comparisons. This table has 5 fields:

`battleId` Integer. A unique, auto-incrementing field which records the order in which comparisons were received.

`winnerId` Integer. The unique ID of the favoured melody in this particular comparison.

`loserId` Integer. The unique ID of the unfavoured melody in this particular comparison.

`musician` Boolean. Indicates whether the comparator identified as a musician or not. Defaults to 'false' (stored as either 1 or 0 in the database).

`time` Integer. The unix time at which the comparison was submitted.

Table: melodies

This table has 11 fields:

`id` Integer. Unique Id indicating the order in which melodies were generated. Also, a melody's generation of introduction can be calculated by

$$\text{generation}(\text{id}) = \begin{cases} 1, & \text{if id} \leq \text{popsize} \\ 2 + \lfloor \frac{\text{id}-(\text{popsize}+1)}{\text{percentageToReplace}\times\text{popsize}} \rfloor, & \text{otherwise} \end{cases} \tag{4.2}$$

`melodyString` A string representation of a given melody, written in the format described in 3.1.2.

`wins` Integer. This field was originally to hold the number of times this melody has been favoured in comparisons within a generation, but is no longer used.

`defeats` Integer. This field was originally to hold the number of times this melody has not been favoured in comparisons within a generation, but is no longer used.

`totalWins` Integer. The total number of times a melody has been favoured in a comparison since the project began.

`totalDefeats` Integer. The total number of times a melody has not been favoured in a comparison since the project began.

`introducedAtGeneration` Integer. The generation at which a melody was introduced into the population.

`parentAId` Integer. The unique ID of the melody which contributed genetic material to the first part of this melody. Used for tracking heredity of melodies in analysis.

`parentBId` Integer. The unique ID of the melody which contributed genetic material to the latter part of this melody. As above, used for tracking heredity of melodies in analysis.

`position` Integer. Dictates the order in which melodies are selected for presentation to a user.

`removedAtGeneration` Integer. The last generation in which a given melody was in the population.

Table: contributors

This table has 3 fields:

`name` String. The contributor's name.

`email` String. Their email address.

`contributions` Integer. The number of comparisons they have made. Recorded by means of a cookie.

4.3.3 The algorithm in practice

This is a description of what happens when a user visits the “compare” page of the project’s site. This page is the user’s entry point into the algorithm, where they can compare pairs of melodies in order to give the system enough data to rank them.

Upon requesting the “compare” page, a function “go” from the PHP script “ga-utilities/melodySelector.php” is called. This queries the database, ensuring that a population of melodies in fact exists, then retrieving the current position counter’s value and incrementing it. The position counter is never reset, but the positions are numbered from 1 to the population size. For this reason, a modulus operation is performed to find the actual position of the next melody to be compared. The other is chosen at random from the remaining population. Note: the position counter gives us the number of calls made to the “compare” page, meaning we can calculate how often users did not make a comparison.

Now we have the position of the next two melodies to be compared against one another (the competitors), we make a database query to get the IDs of the competitors. These will be returned to the script on the “compare” page, and written in to the html to return which melody is preferred. Before that, the “go” function checks that the appropriate sound files exist, and creates them if not (how is explained later). The sound files are named for the melody they are expressing. In other words, the melody with ID 1 will be named “1.mid”, “1.wav”, “1.mp3”. The creation process is quick, and only adds a few seconds at most to page loading time. It only occurs the first time a melody is compared, then the audio file(s) are retained in the “music” folder.

When the user has made a submission, a POST request is sent to the “ga-utilities/melodyChoiceHandler.php” script. On receiving a valid POST request, the SESSION variable is set to reflect that this user has made a contribution. This allows them to leave their details, for credit and consideration for a prize. This will be covered later. The speed of melody playback as specified (or not) by the user is also saved in the SESSION variable. If the user has left their details, then the number of contributions they have made is incremented by one. Otherwise, the number of contributions they have made is stored in the SESSION variable in case they should choose to leave their details later.

The results are submitted as a string in a simple, hyphen-delimited format "winnerID-LoserID". This is split at the hyphen into an array. A database query is then made to ensure that the melodies referred to in the POST are still in the population. This is to counteract a situation where in the time between a user loading the compare page and submitting their results, one or both of the compared melodies has been removed from the population. This could happen, for example, if a user leaves the page open for a day or two before returning to it. Finally, the "battles" and "melodies" tables of the database are updated with the information about the user's choice, along with other information including the time of the query. Finally, the function "checkGenerationThreshold" from "ga-utilities/GenerationFunctions.php" is called, to determine whether or not to create the next generation.

"checkGenerationThreshold" pulls the project parameters "gamma", "popsize" and "percentageToReplace" from the database, and generates a random number n such that

$$n \in [1, \gamma \times \text{popsize} \times \ln(\gamma \times \text{popsize})] \tag{4.3}$$

If $n = 1$, the next generation is created. This function then calculates the number of offspring to create and passes it to the "nextGeneration" function.

The "nextGeneration" function in "ga-utilities/GenerationFunctions.php" is responsible for this. It is in the first database call that balanced rank estimation actually occurs, relying on the database to return the IDs and string representation of that fraction of melodies whose scores, given by $totalWins - totalDefeats$ are in the top $n\%$, where n is the fraction to cross over. This function pairs off these potential parent melodies at random and passes the melody strings them to a utility in the class "GeneticFunctions" (ga-utilities/GeneticFunctions.class.php) to produce offspring. These offspring are initially just the melody string, but there is considerably more metadata which needs to be assigned before introducing the new melodies into the population. At this point, the state of the database has not been updated to reflect the new generation. A database call retrieves the IDs and positions of the bottom $n\%$ by balanced rank estimation, then uses the IDs to make another database call to set their "inPopulation" attributes to "false".

Their positions will be assigned to the new melodies.

All information about the new melodies (see the description of the “melodies” table for details) is collated into objects before being written to the database. These objects are of the “Melody” class (”ga-utilities/Melody.class.php”). The melodies and their metadata are finally written to the database by a method of the Melody class. These objects are analogous to rows in the “melodies” table in the database. These melody-objects are created and written to the database in pairs which share parent melodies. The reason for this is to optimise my code; the effect is that adjacent melodies in the database share parents.

4.3.4 Rendering the melodies

When a melody is required for comparison, a script checks for the existence of the appropriate melody file in an audible format. Converting the melodies from a string representation to a wave/mp3 file involved several stages. Upon determining that audio representations of a melody do not exist, its ID, which will become its filename, and note representation (in which notes are represented by fixed pitch values in an arbitrary key rather than offsets from the previous note) are passed to the “writeMelody” function of the “MelodyGenerator” class (ga-utilities/MelodyGenerator.class.php). To do this, the string representation of the melody is first converted to a 2-dimensional array. The inner dimension are arrays of length 2 representing a note’s offset and duration in that order. The outer dimension is an array consisting of note arrays. Array representations are easier to manipulate programmatically than strings, hence the conversion. This representation is then converted to a fixed pitch version by the “getNotes” function of “MelodyGenerator”. I chose not to abstract this process away behind the “writeMelody” function as “getNotes” takes a starting pitch (in midi values) as a parameter for producing a set of absolute pitch values and I did not want to hide this option.

The “writeMelody” function itself is a wrapper around the Midi package I used [2], hard-coding certain parameters with which this project is not concerned, such as instrument type and note velocity. It allows the creation of midi files consisting of a single track where rhythm and pitch are the only variables. Even speed was an arbitrary choice, as I have provided the user with facilities to change the speed. As such, this function is not particularly versatile, but simplifies the process to meet this project's needs. MIDI velocity for all notes was set to 90. Due to an initial misunderstanding regarding MIDI timebases, I set the beats per minute (bpm) to 35, and the default value of a note to be a quarter of a crotchet (quarter note). This gave the effect of listening to the melodies at 140bpm, which is moderately fast. Using a moderately fast speed was a deliberate choice aimed at further reducing the risk of fatigue. After introducing speed controls, the default speed became less important. I chose not to amend the bpm value, as it worked in a satifactory way as it stood.

After having created and written a midi track named after the ID of the melody it represents, the function executes a system call which creates the wav and mp3 format versions of the track for delivery to the user. The open-source timidity [21] synth creates the former from the midi file, then the LAME encoder [7] creates an mp3 from the resulting wav file. Between these two formats, most browsers are able to play the tracks.

Going via MIDI to create a playable track may seem like an unnecessary step, but midi as a format is much easier to manipulate directly than WAVE, and is inexpensive to process. Leaving the wave and mp3 rendering to pre-existing, tested and dedicated software seemed like a sensible idea.

4.3.5 Overview of Code Structure and Responsibilities

The application is structured as a web app. It is driven by PHP and uses MySQL to store the state and the global parameters of the project.

The root folder, "mutunes/", contains the user-accessible pages of the site, and the utilities and scripts which power them are found in the following directories

Directory: components

The components of the page to be rendered and served by PHP. Items in this directory include items like the menu and footer. They are unremarkable components, dynamically inserted into pages to avoid verbose repetition of code and mark-up.

Directory: php_utils

For general useful utilities, but now only holds a function for querying the database (function "query" in "db_query.php".

Directory: ga-utilities

The classes and scripts which power the Genetic Algorithm. The role of these files was explained in more detail in "The algorithm in practice", but in summary, these files are:

Script: convergenceData.php

Script: Gaussian.class.php Contains the "Gaussian" class, responsible for generating parameterised Gaussian distributions. Originally I had several purposes for it, but ultimately it was only used for generation of the initial population. I could not find any pre-existing code which fit my needs, driving my decision to write to own class for Gaussian operations.

Script: GenerationFunctions.php Functions involved in and responsible for creating the next generation.

Script: GeneticFunctions.class.php Utility class. Genetic operators. No longer used.

Script: GeneticFunctionsWithDuration.class.php Utility class. Genetic operators capable of manipulating a melody representation which also encoded note durations.

Script: melodyChoiceHandler.php Script to process user's choice of preferred melody and metadata about the user and algorithm in general.

Script: melodyChoiceHandlerTest.php Script to process autonomously-generated data used to test that evolution was actually occurring and that my implementation was functioning.

Script: Melody.class.php Data class which reflects the way the melodies are stored in the database.

Script: MelodyGenerator.class.php Utility class for generating random melodies.

Script: MelodyGeneratorWithDuration.class.php Utility class for generating random melodies, representing pitch and duration.

Script: melodySelector.php Script responsible for selecting and presenting a pair of melodies to the user. It also has to make sure the relevant sound files exist.

Script: midi_class_v178 Pre-existing midi class [2] found online. Capable of a great deal more than I used it for, but sufficiently fast and easy to use.

Script: player.php Function responsible for rendering the audio player in the "compare" page to play a given melody.

Script: populationGenerator.php Script which generates the initial population according to the specified parameters.

File: testlog.txt

Script: Vector.class.php

Directory: javascript

Scripts containing the javascript functions active on the main page.

Directory: music

Holds the midi, wave and mp3 files after they have been created.

Directory: styles

Stylesheets for the website.

4.3.6 Coding paradigms, practices and layout

The human element in this project was the major bottleneck for speed. The speed of execution was irrelevant to the success of this project so long as it did not discourage human participation, so writing extremely fast, efficient code took second place to clarity and good abstraction. Most of the scripts are written according to the imperative paradigm, which makes following the flow of events easy to follow. I tried to follow general good practice such as frequent comments and meaningful variable and class names. Some object orientation is used to group semantically-related data and methods into utilities or useful containers for data which reflect some aspect of the database structure. The files containing class definitions are indicated by ".class" in the file name.

General project parameters are not hard-coded, but rather stored in the database and manipulable by an administrator (me) from the "cms" page. This offers flexibility were the project to be run again. The state of the project is completely separate from its implementation, being stored in the MySQL database. This was necessary in order to be able to analyse the data fully. As a result, most of the PHP functions involved with the Genetic Algorithm itself are little more than wrappers for parameterised database queries.

4.3.7 Difficulties and Improvements

The finite amount of time I had to develop the code which powered both the site and the algorithm, which began after a period of design, was eaten into by my attempt to use the ill-supported weblocks [4] framework. This time was eroded from the other end by the fact that the project needed weeks to run to maximise the amount of participation, and once the site (and thus my code) went live, making anything but minor changes was potentially very dangerous. It would also compromise the validity of the experiment. I learnt this to my cost after trying to fix a bug. I introduced an error which was not immediately obvious and accidentally ceased the collection of data for a few days. While it is impossible to determine precisely how much input has been lost, figure 5.1 suggests that the time when this occurred was when participation was relatively low.

Despite time limitations, I believe that my implementation clearly implements my algorithm in a flexible, well-encapsulated way. Most importantly, by every means I can measure it, it worked. I believe it will make a strong basis for future iterations of the project. In future versions, I would prefer to move to an object-oriented database and object-orient the rest of the code generally. Although a relational database served perfectly well, the melodies as well as their metadata (from here: melody-objects) lend themselves to object orientation. Certain data about them (e.g., their balanced rank estimation score) is a function of other data, and some data aught to have access to them protected in the ways that object orientation facilitates and encourages.

4.4 Testing

In order to test that ranking by pair-wise comparison really did influence the evolution of the population, I copied the "compare" page of the site and modified it to select the melody with the higher level of disjunctness. If the two melodies were equally disjunct, the winner was selected at random. The disjunctness of a melody was calculated as an average of the magnitude of offsets between notes such that

$$\text{disjunctness}(\text{offsets}) = \frac{1}{|\text{offsets}|} \sum_{x=1}^{|\text{offsets}|} abs(\text{offsets}_x) \tag{4.4}$$

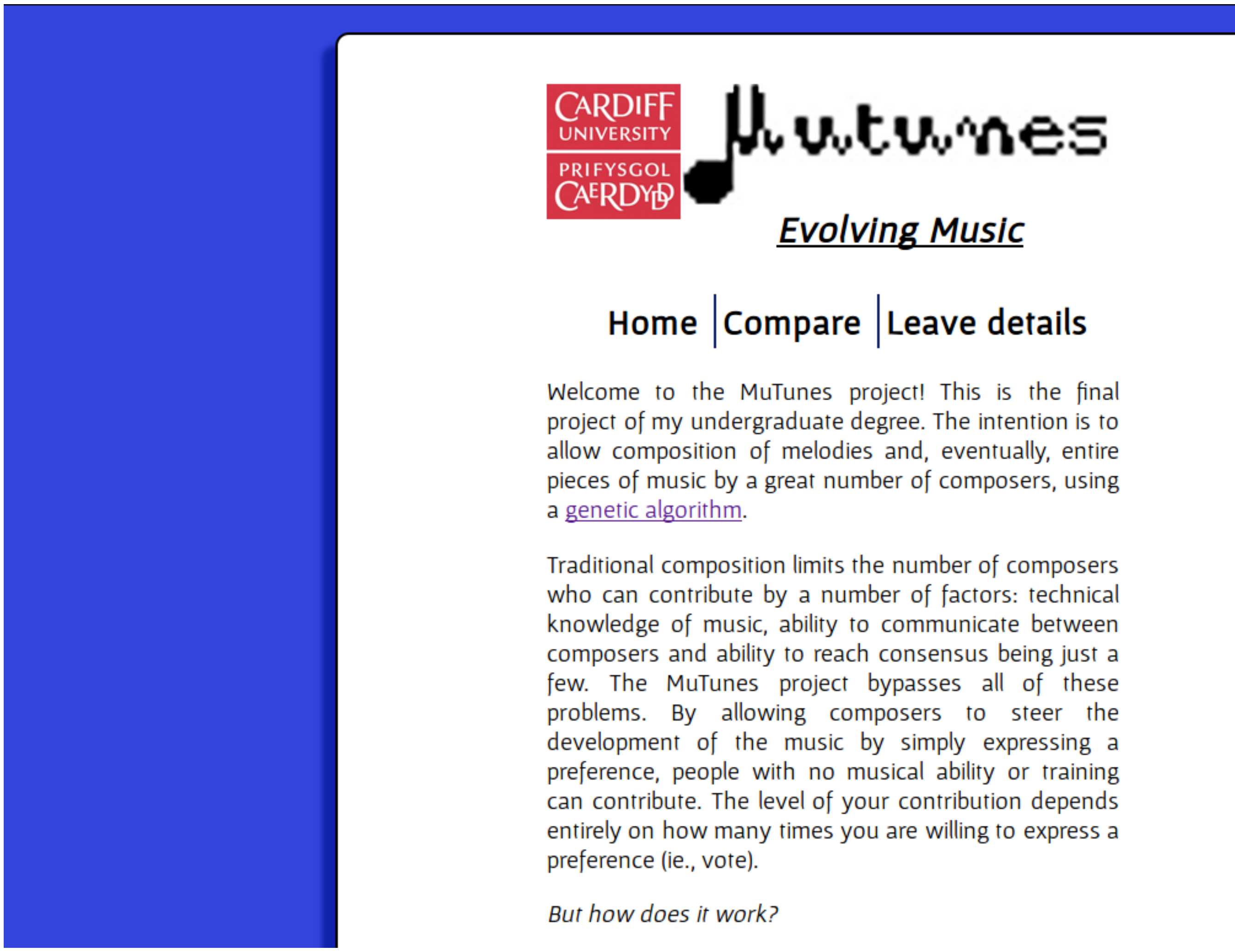
As the melody representation stores offsets, this was straightforward to calculate. Note duration was not taken into account. If the evolution was being swayed as it should, subsequent generations should be increasingly disjunct. In theory, this function has the mapping disjunctness(offsets) $\mapsto [0, \infty)$, but the limits placed on the initial generation of melodies means that this figure will probably not exceed the teens. I ran this experiment with the same general parameters as the main experiment. Figure 4.4 shows that the disjunctness did indeed rise, confirming that evolution is being swayed by the selections made by the "user" (in this case an autonomous script). In this figure, disjunctness is being measured as the mean offset between notes in a melody. This measurement of disjunctness is poor. It should take into account the modal average offset between notes, rather than the mean. However, the mean was selected for and so the mean I have measured. There is no reason this measure is not sufficient to show that evolution is occurring and the system is working as expected.

4.5 Launch, Progress and Promotion

Public promotion of the project began on March 14th of this year. The project was announced using emails from the school of Computer Science and various other forms of social media. Being freely available on the web, there was the potential for knowledge of the project to spread virally. Social media also gave me a platform to communicate with people who expressed interest in the project. I staggered announcements about the project so I could measure the receptiveness to the project in different forums. I had predicted that the time immediately after advertising μtunes would see large spikes in participation, and by plotting a histogram (Figure 5.1), my prediction was proven correct. The largest spike was on the link sharing site reddit. Reddit is divided into various special interest "sub-reddits", identifyable in URLs by the prefix "/r/". I submitted links to μtunes to both the /r/music sub-reddit and the /r/programming sub-reddit. The latter posting preceded a short period of time during which almost two-thirds of comparisons were made, making it reasonable to assume that I owe /r/programming a debt of gratitude!

To incite interest, and encourage people who may only have participated minimally to participate more, I advertised two prizes for the participants who undertook the most comparisons. These were a harmonica and a Raspberry Pi. It is not clear the extent to which this made a difference, but I felt it was a worthwhile investment to try to diminish the risk of low participation. In general I made an effort to emphasise those aspects of the project which a non-scientific audience may find interesting, such as being able to contribute to a piece of music with no musical ability.

In order to maintain interest in the project, I recorded versions of the (then) top ranked melody in different styles - a heavy metal version and a simple piano arrangement. It is unclear whether this generated any more interest in the project, but it does help demonstrate and clarify the fact that the melodies were improving and of some musical merit. It was also very enjoyable, and a practical way to remind myself of the point of the project and keep my own enthusiasm up. The particular melody I picked (ID 127) was entirely diatonic, conforming to a melodic minor scale, with it's first note being the 2nd of the scale.



Welcome to the MuTunes project! This is the final project of my undergraduate degree. The intention is to allow composition of melodies and, eventually, entire pieces of music by a great number of composers, using a genetic algorithm.

Traditional composition limits the number of composers who can contribute by a number of factors: technical knowledge of music, ability to communicate between composers and ability to reach consensus being just a few. The MuTunes project bypasses all of these problems. By allowing composers to steer the development of the music by simply expressing a preference, people with no musical ability or training can contribute. The level of your contribution depends entirely on how many times you are willing to express a preference (ie., vote).

But how does it work?

Figure 4.1: Homepage of the $\mu Tunes$site.

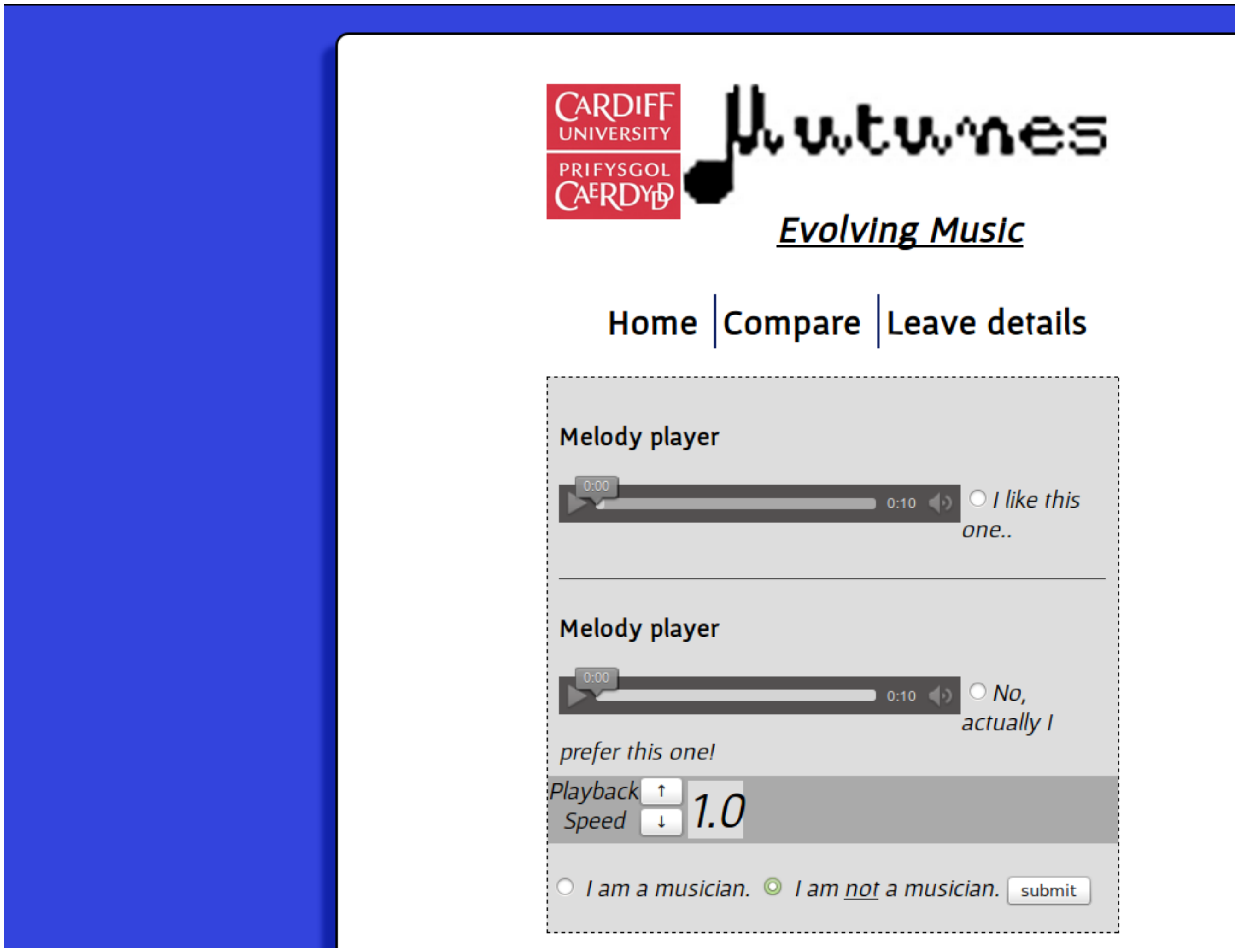
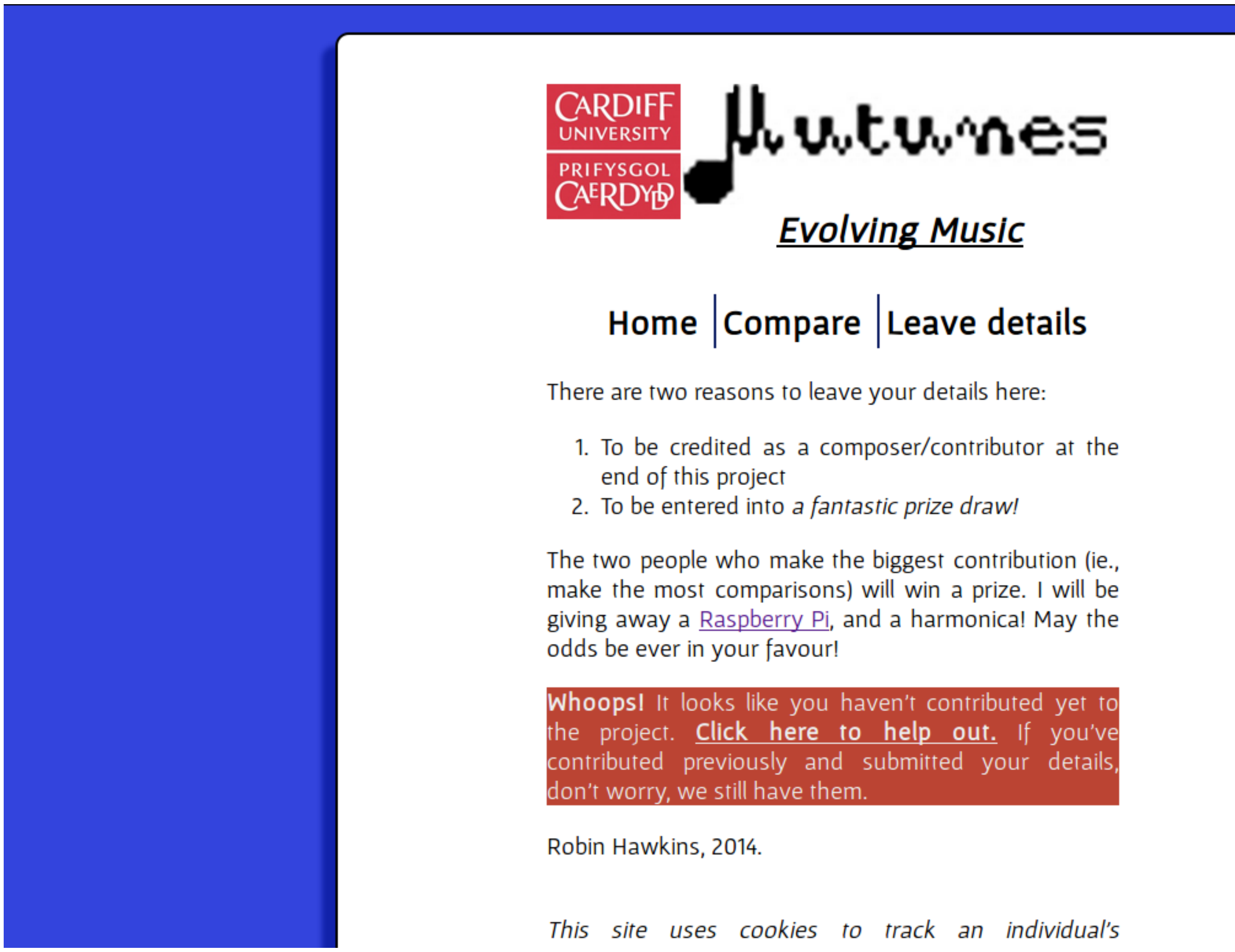


Figure 4.2: Comparisons page of the $\mu Tunes$site.



There are two reasons to leave your details here:

1. To be credited as a composer/contributor at the end of this project
2. To be entered into *a fantastic prize draw!*

The two people who make the biggest contribution (ie., make the most comparisons) will win a prize. I will be giving away a Raspberry Pi, and a harmonica! May the odds be ever in your favour!

Whoops! It looks like you haven't contributed yet to the project. **Click here to help out.** If you've contributed previously and submitted your details, don't worry, we still have them.

Robin Hawkins, 2014.

This site uses cookies to track an individual's

Figure 4.3: "Leave details" page of the $\mu Tunes$site, denying access as no comparisons have been made in this session.

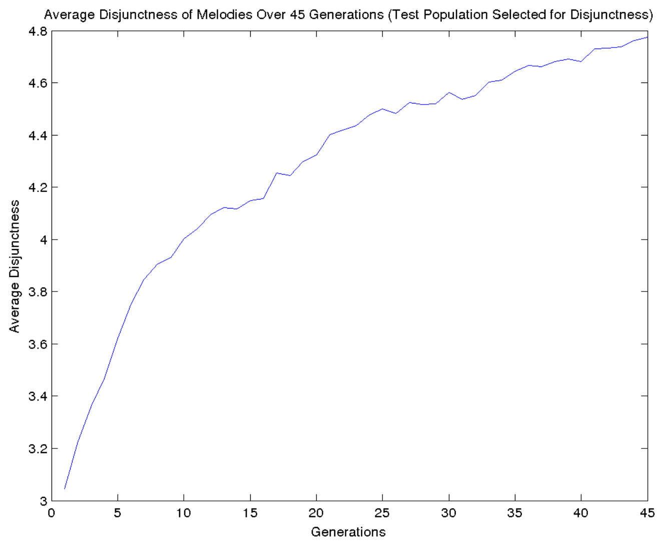


Figure 4.4: Frequency of Particiation Over Time

Chapter 5

Results and Evaluation

Throughout this chapter I will make reference to melodies by their ID number. These melodies can be listened to on the $\mu Tunes$ project site at listen.php.

Analysis was performed in matlab. I have included all data a scripts that I used with this report. The functions and scripts are commented and named in a manner which (I think) is self-explanatory.

5.1 Participation

One of the problems I had predicted was lack of interest and minimal participation. On the contrary, in total the number of comparisons made exceeded my estimate by multiple times. People were driven by scientific curiosity and musical interest, or in some cases, both. In this respect, this project was a success, a lot of data has been gathered for analysis, and responses were interested and positive.

I can analyse participation in two ways:

5.1.1 General response to promotion

As I recorded the times at which comparisons were made, accurate to within a few seconds, I can show that rate of participation increased after advertising the project in different forums. As I staggered these advertisements, I can also show which forums provided the most participation, and thus speculate why. They are listed chronologically by date I first used the medium to promote the project. I should note that I added the functionality to record the time of a comparison three hours after I first promoted the project. Comparisons made prior to that time have had their times linearly interpolated over those three hours. It's hardly even visible on the graph.

Originally, I did not intent to stagger the promotion across different social networks by any significant length of time, as I wanted to accrue as much data as possible as fast as possible. When I saw the rate at which users were participating after the first two announcements (Facebook and /r/music), I then made the decision to stagger any remaining promotion to try to asses what was and was not successful.

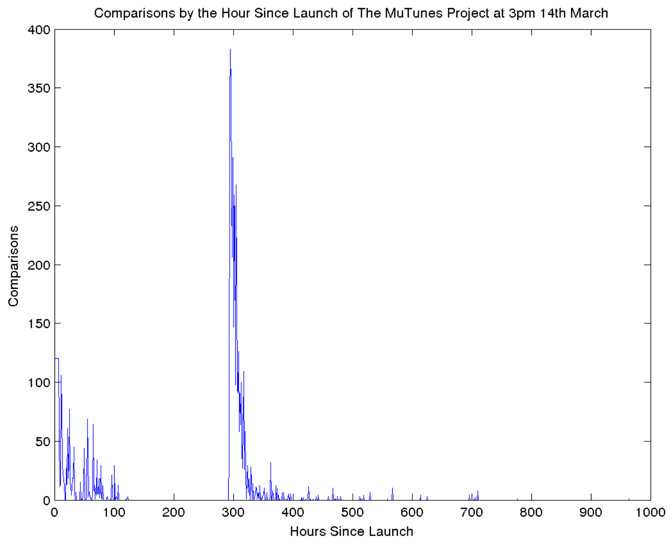


Figure 5.1: Frequency of Particiation Over Time

Email March 14th. I was not responsible for sending these emails, so I have no times for them. I speculate that their reach was not large compared to some of the social networks used to promote the site.

Facebook March 14th, 3:09pm. This marks the beginning of the first large spike in participation in the project. Figure 5.1 shows that participation was strong from the outset.

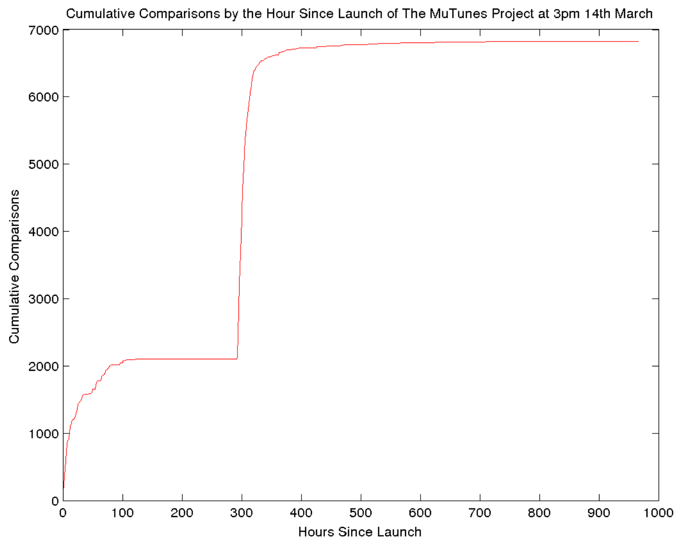


Figure 5.2: Cumulative Frequency of Particiation Over Time

Reddit: /r/music A community interested in sharing good and interesting music. March 14th, 9:21pm. A score of 8 (scoring dictates the prominence of content and is determined by the number of "upvotes" minus the number of "downvotes" given by other users), indicates little interest from here, suggesting we can attribute most of the traffic at around this time to Facebook.

Twitter First tweet made on March 26th, 12:02pm, to 645 followers.

Reddit: /r/programming A community interested in the practice of and culture around programming. Post announcing project made March 26th, 12:10pm. The very large spike commencing just before 300 hours coincides with the time at which this post was made. Although the spike could also be attributable to traffic from twitter, the lack of responses from there and the reddit score of 145, compared to 8 in the music subreddit, suggests that this is the source of the traffic.

Summary of participation

The graph shows two spikes. The first spike - shorter and broader - is likely due to traffic from Facebook, and the second - taller and thinner -

likely due to the reddit's /r/programming. The latter generated almost double the traffic than the former, in a shorter space of time (4000 comparisons overnight). This suggests that targetting special interest groups is a more effective method of driving participation than general social networking amongst friends and peers. Advertising to the latter also runs the risk of being perceived as spam, which would discourage people from participating.

5.1.2 Individual behaviour

I used cookies to track the level of participation in the project of those people who had left their details. This was to decide who to award incentives too, but it has had the side-effect of allowing some analysis of how much the typical user participated.

102 participants left details. On average, each of these individuals undertook 20.3 (1 dp) comparisons. This is misleading, as a small handful of people actually did most of the comparisons. The total number of comparisons was 2027, but ignoring the top 5%, who comprise these extreme outliers, we are left with 971 comparisons at a mean rate of 10.1 (1dp) comparisons. Determining why the top 5% contributed so much would be extremely useful to people trying to undertake massively crowd-sourced projects. They constituted more than half the comparisons recorded for this group. Assuming this information holds true across all participants, we can estimate the number of unique participants from the number of comparisons. At time of writing, this figure stands at 6,843. Knowing that the average number of contributions amongst those who left their details is 20.3137, we can estimate the number of unique contributors at $6,843/20.3137 = 336.8663$. As using floating-point numbers to count people does not make sense, let us round this to 337 for convenience. I suspect that this is an underestimate - those who left their details seem likely to be more invested in the project than users who didn't, and so would have made more comparison. I think there were probably more individuals making fewer comparisons, so 337 should be treated as a lower bound.

I will not divulge personal information here as a matter of good practice, but as I know one of these super-contributors, I was able to ask them what encouraged them to be so involved. They said that they, as a musician, used to be interested in improvisational jazz, which many of the generated melodies reminded them of. The individual in question is also interested in the scientific aspects of the project. I asked for a quote in a Facebook chat:

> *μTunes*: Hi! I was hoping to grab a quick quote from you - what made you keep coming back to do comparisons on my project?
> G: Sure, a couple of things; one the lure of a free harmonica, two I quite liked the music and it threw up some interesting 'free' music
> G: Very atonal and interesting sounds
> *μTunes*: Cool, thanks
> G: no worries
> *μTunes*: So the novelty of the music appealed, rather than thinking it was "great", I suppose?
> G: Yeah, though I haven't listened to it recently so am not sure how they sound atm (presumably better)
> G: I quite like the questions it arises too, is it music? Does music have to have meaning or can it be generated? Is it art
> G: ?
> *μTunes*: It's got some features of music, that's for sure.
> G: but are features enough? can music be made with no motive or meaning? It's quite an interesting thought

Although the sample data is tiny, it is probably not to unreasonable to suggest that strong enthusiasm to participate in a project is more likely in an individual the more the project appeals to their existing interests. This uncontroversial notion is further supported by the response from reddit's /r/programming subreddit. However, it is difficult to say whether a few were contributing repeatedly, or many were contributing a little. I might go further afield and note that in both cases, an interest in programming is given, which can suggest an obsessive personality type.

5.2 Evolution and Increase in Musicality

In order to establish a basis for comparison and a baseline against which the results must be measured, I ran a parallel experiment in which I replaced the human element with noise. This baseline experiment was seeded with the same 100 melodies which were the original population of the measured results, and shared the same parameters. However, comparisons were made on a random basis, with the odds of either melody in a given comparison being chosen being 0.5. The measured results should demonstrate more musicality on a generational basis and stronger trends towards musicality intergenerationally. I have devised a number of metrics by which to measure musicality, and detailed the process of calculating them with the results. Other metrics were developed by my supervisor, Dr. Kirill Sidorov, for our paper [11] based on this project. I am reusing some of his analyses with his permission.

There is almost no limit to the ways in which you can analyse a piece of music, even when it's representation is restricted just to relative pitch and relative duration. I show measures that demonstrate the success and interesting features of the project. There are undoubtedly more ways I analyse, but there are

5.2.1 Evolution of Melodies

By plotting edit (Levenshtein) distance into Euclidean space, we can see how the melodies diverge over time (figure 5.3). Two distinct evolutionary directions become apparent. All the melodies along these directions share common ancestors. The distinct, clear nature of these directions indicates that just a few melodies are having a disproportionate influence on the evolution of the population as a whole, and are likely considered much better than most others to be having this influence.

5.2.2 Hierarchy of convergence

I have listed indicators of emerging musicality in order of granularity, from most obvious to most subtle and sophisticated.

General reduction in entropy

Some reduction in entropy over the generations is expected, as for each generation, 20% of the information is lost and less is put back in. This is often the nature of convergence in Genetic Algorithms. To indicate any convergence, we would expect to see a greater reduction in entropy in the measured results over the control.

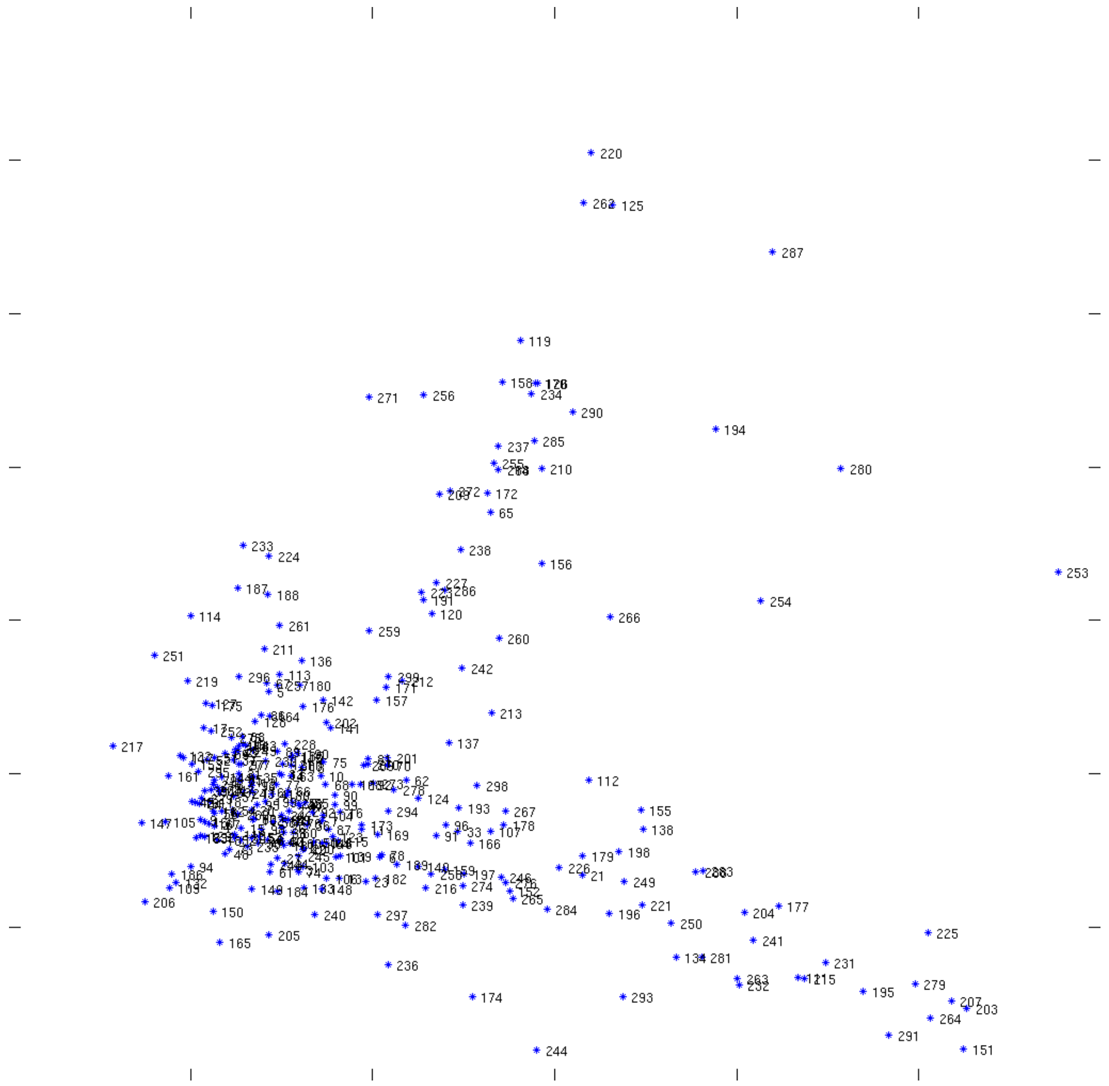


Figure 5.3: Edit Distance of All Melodies from One Another

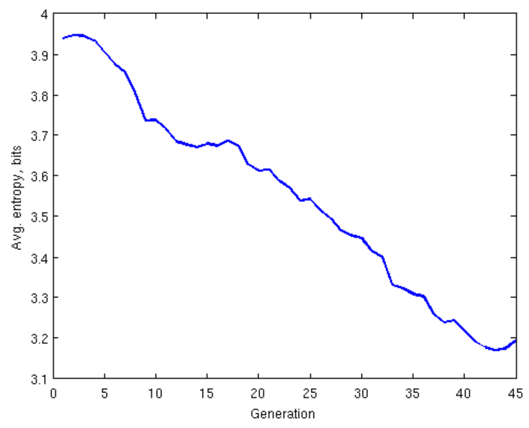


Figure 5.4: Average Entropy in Bits Over 45 Generations (produced by Dr Sidorov)

Figure 5.4 shows a measurement of Shannon's entropy for both the rhythm and melody of the real and control results, such that

$$\text{Entropy}(X) = -\sum_i P(x_i) \log_2 P(x_i) \tag{5.1}$$

It is clear to see that something greater than just the loss of information as a side-effect of creating new generations is going on here. Already we can conclude that the measured results are being strongly driven by the data gathered from pair-wise comparisons.

Rhythmic Entropy

Figure 5.5 demonstrates a rise in entropy of the rhythms over the generations. This would be indicative of strong selection for more interesting rhythms, but I am unable to rule out the possibility that the genetic operators used have the side-effect of introducing more rhythmic variation as the generations progress. The control experiment certainly indicates this. Despite that doubt, there is a clear rise in the rhythmic entropy in the measured results over the control. This strongly suggests selection for more complex rhythms. Combined with the fact that several melodies in the top 10 overall melodies contain short trills and grace notes, I think this is a reasonable conclusion.

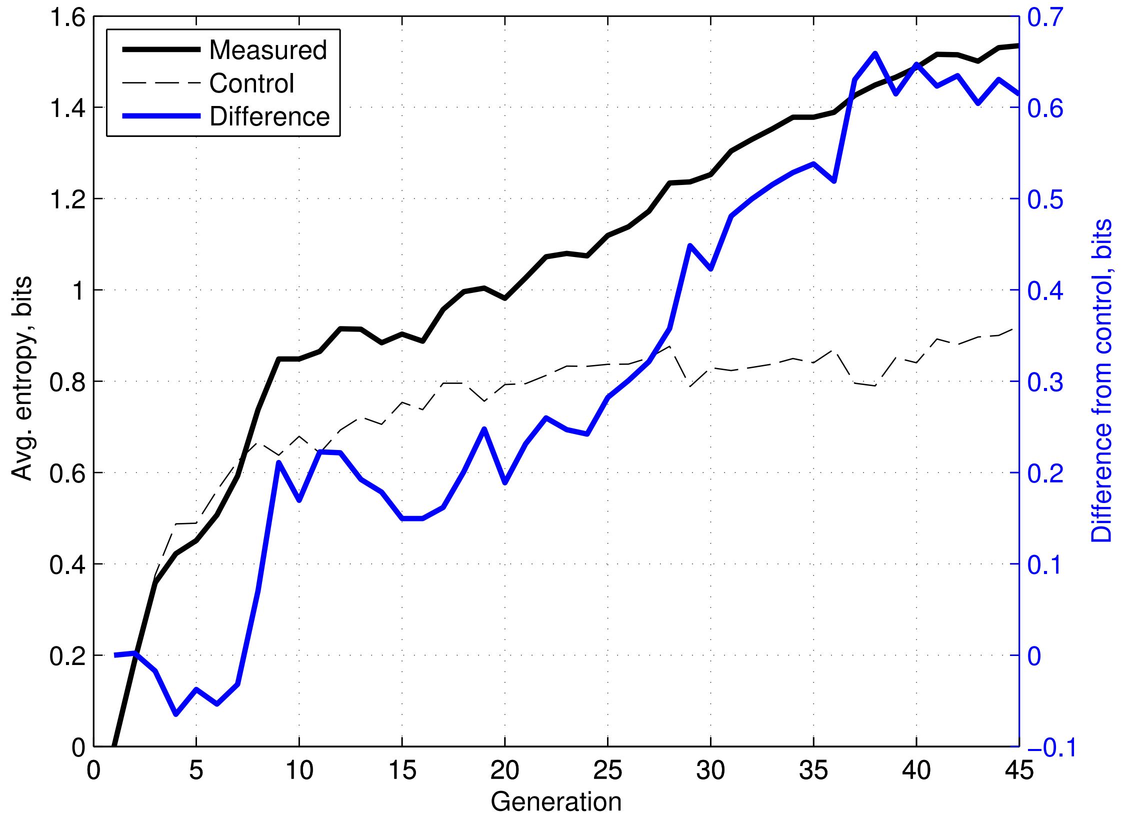


Figure 5.5: Entropy in the Rhythm of Melodies over 45 Generations (from Sidorov 2014 [11]

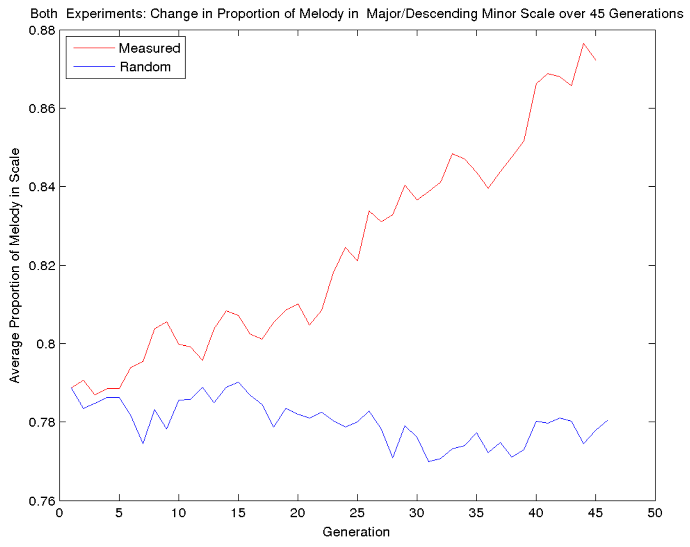


Figure 5.6: Comparison of diatonicity between random data and measured data

Rise in diatonicity

Diatonicity is the adherence of a melody to a traditional Western major or minor key. Owing to my melody representation, the melodies are divorced from fixed pitches, so I will be treating key as a pattern of notes. There is nothing that means that the first note of the melody must be the tonic of the scale, so I rotated a mod-12 version of the offsets of each melody through all possibly keys and compared them against a scale. The diatonicity value at each rotation is the number of notes in the melody which fit the current scale over the number of notes in the scale. I returned the maximum of this value. I tested for adherence to major (figure 5.6), harmonic minor (figure 5.8), and melodic ascending (figure 5.7) and descending minor scales (figure 5.6), and took the average over each generation, for all generations. The results for major and descending melodic minor are identical, as they follow the same pattern (just starting at different points).

Across all of these measurements, diatonicity increased significantly in the measured results. The random results did demonstrate a slight trend away from diatonicity, but were, unsurprisingly, fairly random and without a clear trend. The distinction between the two is striking, and amply demon-

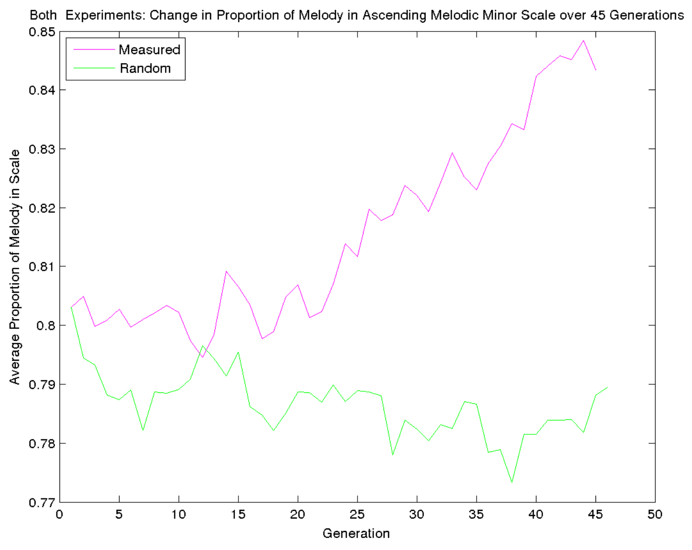


Figure 5.7: Comparison of diatonicity between random data and measured data: ascending melodic minor scale

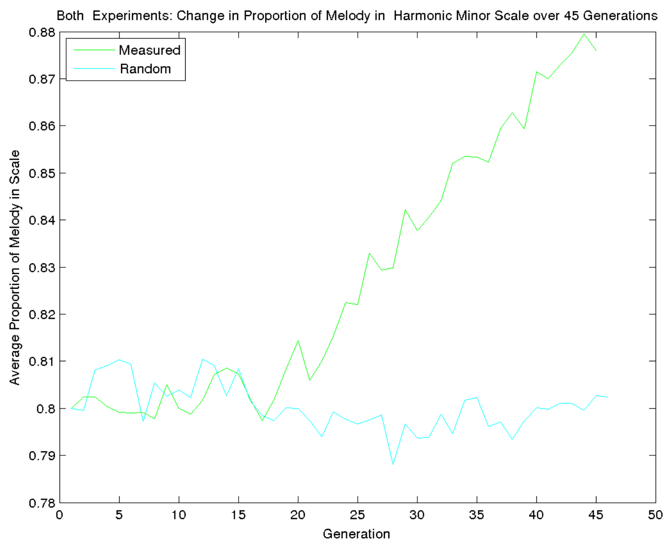


Figure 5.8: Comparison of diatonicity between random data and measured data: harmonic minor scale

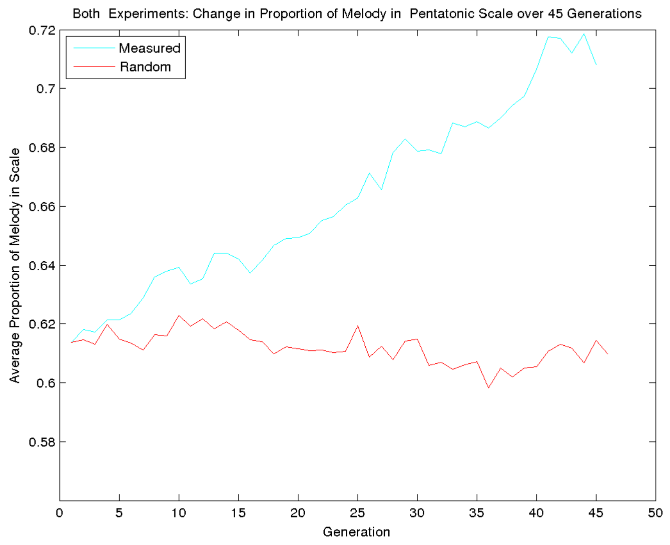


Figure 5.9: Comparison of diatonicity between random data and measured data: pentatonic scale

strates a strong movement towards musicality. Of the measurements, the pentatonic scale stood apart as being less present than the others, but this is hardly surprising as it constitutes a subset of major and descending melodic minor scales.

This removes doubt that evolution is being directed by our human participants, and towards traditional Western diatonic scales.

It is worth mentioning that microtones such as used in scales prevalent on the Indian subcontinent were not used in this project. This was a limitation of using the midi format to render melodies.

Emergence of major and minor keys, frequencies of notes

The measure of how many notes in a melody conform to the same scale is not as powerful a musicality indicator as how many *consecutive* note in a melody conformed to a scale.

Use of the notes of a scale is possible without a melody sounding much like it belongs to a given key. As such, the above measure does not tell us a

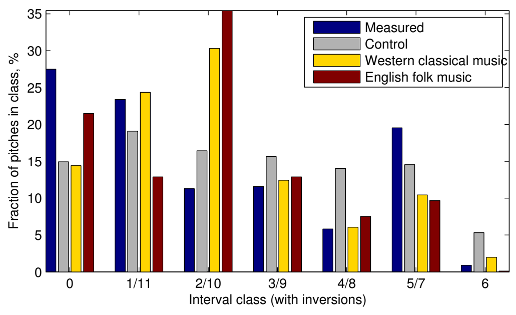


Figure 5.10: Histogram of Interval Classes of Last Generation Compared with Western Classical and Folk (from Sidorov 2014 [11]

great deal about the musicality of a melody. To emphasize my point, consider that a cat could easily walk across a piano in the key of C (white notes only). A melody's belonging to a certain scale is cemented in our perceptions by the frequencies of certain notes from that scale, notably the tonic, mediant and dominant.

Figure 5.10 shows that the melodies of the last generation strongly favour the tonic and the dominant of the scale, especially compared to classical and folk. I believe they have been driven this way because we are dealing with single melody lines, rather than two or more parts which would provide a richer harmonic context. Because these melodies had to sound musical on their own, frequent repetition of the tonic and dominant notes replaced other parts in establishing a harmonic context. This is an effective way to derive a sense of key from a single melody line. Also, it show analytically that the melodies have a strong sense of key, rather than the "cat on a piano" arbitrary collection of notes about which I had speculated.

Narmour's gestalt principles: registal direction and intervallic difference

These are two ideas discussed in Gratchen et al, 2004 [8] from Narmour, 1992 [18]. There are, in fact, five principles which constitute the implication/realisation model, but two are particularly relevant, useful to us and easy to analyse. The remaining two are based on user's perceptions, which I am not privy to, and the other (*proximity*) is not so easy to implement, doesn't give us much useful information and may be skewed by my process of initial population generation.

The principle of *registral direction* states that a small interval implies another interval in the same registral direction, but a large interval implies a change of direction. For example, for consecutive notes $n1, n2, n3$, $pitch(n2) > pitch(n1)$ implies that $pitch(n3) > pitch(n2)$. The principle of *intervallic difference* states that small intervals indicate more intervals of roughly the same size (plus or minus 2 semitones) if the registral direction changes, and maybe slightly larger if it doesn't (plus or minus 3 semitones). Large intervals imply smaller intervals. In both cases, an interval of 5 or fewer semitones is considered small, and an interval of 7 or more semitones is considered large. The principle of *registral return* describes how archetypal a pattern is. An archetypal pattern is one where in a series of 3 notes, the third note is within two semitones of the first. *Registral return* is not really an implication, but is interesting property of the melodies to measure nonetheless. The extent to which the implications of *registral direction* and *intervallic difference* are realised is testable. I calculated a measure of implication realisation as the percentage of implications which were realised, calculated the average for each generation and plotted them over the generations.

Interestingly, I believe I have already demonstrated my own adherence to these expectations. In particular, the ideas that a change of registral direction is implied by a large interval, and that a large interval implies a small one are inherent in my method for generating the initial population; I did not allow the melody to wander more than one octave from the start note. While this situation could arise from a series of small intervals in one direction, I had a series of large intervals in mind.

Both registral direction (figure 5.11) and intervallic difference (figure 5.12) realisations show increases over the generations, both as a trend and over and above the randomly generated data. This is particularly exciting as these measures are much more subtle than the others. As the graph shows, the extent to which intervallic difference implications are realised is quite high even in the initial population. This may confirm my suspicion that I have introduced bias by restricting how far the melody could wander in the initial population. However, the effect of user influence is still evident and visible in the graph; intervallic difference in the measured data *continues to rise* whereas as the random data falls. It is being positively selected for. Realisa-

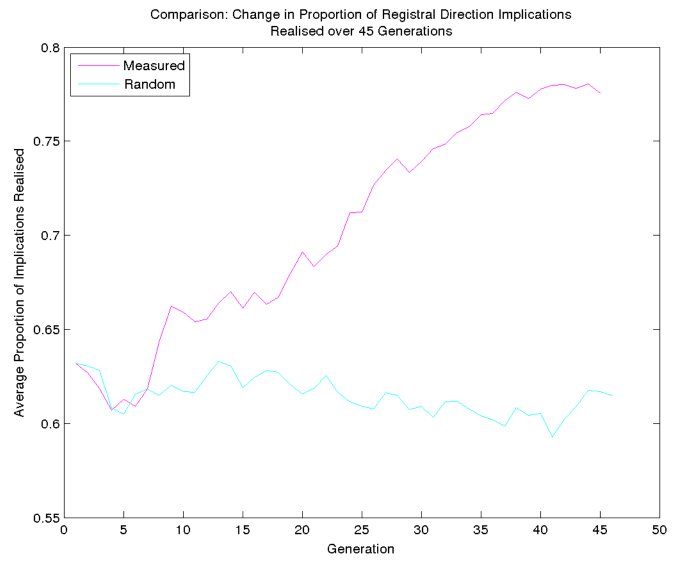


Figure 5.11: Comparison of frequency of realisation of implications of registral direction

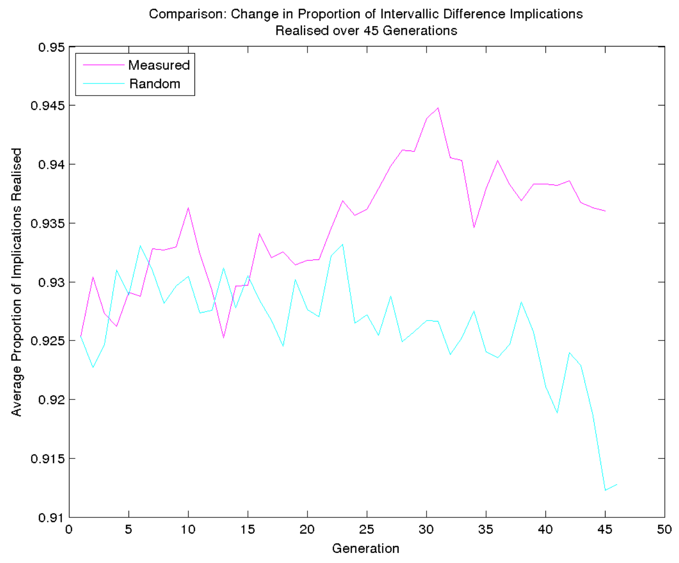


Figure 5.12: Comparison of frequency of realisation of implications of intervallic difference

tion of registral direction implications starts lower but also rises dramatically while the random data falls off.

The significance of these results is clear: the realisation of these implications is perceived as highly musical and they have been selected for strongly. This is significant information. Even as bare as the melodies are, they conform strongly to well-defined rules. This suggests that the rules of Narmour's implication/realisation model constitute knowledge which could be folded into future software to aid composition.

5.2.3 Examples of Music, Comparison to Other Music, and My own interpretation

The top ten highest-scoring melodies of the entire project are 614, 617, 555, 271, 356, 273, 473, 494, 475 and 573 (listed by ID from tenth to first). They are amongst the materials submitted with this report. Listening through them, it is immediately clear that they are closely related - common themes are abundant. Some are so similar that they could be considered theme and variation. Eight of these share melody 39 as a common ancestor, and nine share melody 65 (see Figure 5.1). Perhaps unsurprisingly, 39 and 65 have the most descendants (see figure 5.2. While monitoring the progress of the project, I noticed aspects of melody 39 in many, many others.

The top ten melodies most selected (as distinct from highest scoring) by musicians are 494, 220, 170, 175, 92, 475, 208, 39, 125 and 127 (listed by ID from tenth to first).

The top ten melodies most selected (as distinct from highest scoring) by non-musicians are (by ID from tenth to first) 170, 192, 271, 573, 473, 356, 392, 151, 194 and 273.

Musicians and non-musicians seem only to agree on melody 170. The non-musicians' top 10 has more in common with the top ten by score. Given that the proportion of contributors who identified as musicians is 47.08%, this is somewhat surprising.

Ancestor Melody	Melodies in the Top 10 with this Ancestor
96	1
127	1
153	1
214	1
271	1
273	1
392	1
494	1
95	2
356	2
76	3
175	3
88	5
94	5
103	5
192	5
81	7
119	7
125	7
39	8
65	9

Table 5.1: Common Ancestors of Melodies in the Top 10

Melody by ID	Descendants
103	296
88	298
23	302
94	302
95	314
125	412
119	526
81	528
65	598
39	640

Table 5.2: The Top 10 Melodies by Number of Descendants, from 10th to 1st

Chapter 6

Future Work

6.0.4 Genetic Operators: general improvement

Many of the genetic operators are naive in the way they function. This was a deliberate choice, as the use of anything more esoteric and specialised would have been based on speculation and increased the complexity of the project dramatically. However, feedback from users suggests that more control over what part of a melody is passed on would be a good thing. One-point crossover is a tool ill-equipped for such a task. Also, melodies often had single dissonant notes weakening their musicality, and these cannot be corrected in a single operation as the system stands. For these reasons, thought needs to go into how to further improve and adapt the operators to the task in hand

The rhythmic changes in the melodies arose as a product of one-point crossover. This rendered rhythm little more than a capricious side-effect of the process. Rhythm has been treated as secondary to melody in this project. It is not clear from analysis whether or not rhythmic variation is correlated to popular melodies.

We have now demonstrated that the populations become increasingly musical as the generations progress. Thus we can infer that the refinements required to increase musicality become finer and finer as the generations progress. In the current system, the mutation operator affects not only one note, but all the notes after it by effectively pitch-shifting them. This is because is operates in offset-space rather than pitch-space. When the melodies have low musicality, this unsubtle approach is useful as we need to search a broader area for solutions. However, it threatens the musicality of quite good melodies by making broad changes. I propose introducing a second operator which truly works on a single-note basis. In a manner similar to simulated annealing, the single-note mutator would be employed increasingly throughout the project. It could be used with a probability inversely proportional to the reduction in entropy of the population.

6.0.5 More complex music: dynamics, harmony, and beyond!

The simplicity of the music that has emerged from this project is an artificial constraint. Although this was necessary to expedite evolution within the time available to me, in future efforts should be made to expand to encompass more areas of music.

The melodies produced by *μTunes* have a strong emphasis on certain notes of the scale in order to provide their own musical context. This is because they have had to work well in isolation, which, for melody lines in most forms of music, is an abnormal situation. It is likely that simple melodies which sounded insipid or overly repetitive would have been comparable to much modern pop music with the appropriate chords accompanying them. Instead they were overlooked and removed from the population. Even by underpinning a melody with simple root notes, we can change the impact it has on us dramatically. For this reason, I think a means of evolving chord patterns (perhaps consisting of only the root notes) to accompany the single-note melodies should be the next logical step for the project.

Naturally, raising the level of complexity of the music we want to get out of the system raises the complexity of the information we need to put into the system. One of the strengths of the system is the ease with which people can interact with it. This ease should not be sacrificed lightly. Consideration of how to achieve this merits another project in itself. The simplest option I can think of is to select the most popular melodies from the current iteration of the project and evolve harmonies to compliment them. This would considerably lengthen the process of composition, though. Another approach might be to have a fixed set of harmony parts which do not evolve (or even if they do evolve, the first generation could be seeded with well-known chord progressions). Originality in harmony parts is not as crucial to a piece as originality in the melody line; I refer you to “4 Chords”, by Axis Of Awesome and a story about Pachelbel’s canon as anecdotal evidence. When melodies are presented, they would be presented accompanied by an arbitrary member of the harmony set. As the harmony would effect the perceived musical value of the melody, we could then use the rankings to determine which harmony part fit a given melody the best. Even if a melody died, the harmony information may be somewhat applicable to any living relatives.

The fact that our current representation is as a set of offsets gives twelve possible harmonic relationships of melody parts to harmony parts. It would be possible to have the least dissonant relationship automatically detected. Here we have a trade-off between user autonomy and ease of use. Harmony/chords provide a context which may allow knowledge-based corrections to occur to the melodies. As they often imply a key, it should be abundantly possible to apply an autotuning algorithm to a melody so it fits better with the harmony, without disrupting the contours and general pattern of intervals which give it its unique character. The problem here is that we are prescribing a key. This could be a function which only effects the phenotype and not the genotype, allowing a more “natural” evolution (this is not an example of the naturalistic fallacy; I mean “uninterfered with” rather than natural).

It might be tempting to incorporate the evolution of overall structures of a piece into the system, and although it is important, I would not prioritise it. This is because $\mu Tunes$ set out to make composition achievable for almost anybody, and so it’s focus should first be on simplifying those areas which require the most skill to engage in. Deciding the number and order of sections in a piece is easy *in comparison to actually writing those sections.* If not easy, then infinitely more accessible to the lay person. Likewise, repetition is so prevalent in so much music that simply repeating the melodies we have evolved (perhaps with some small variation, of which we have examples) is sufficient to begin building a complete piece of music. Similarly to structure, dynamics are much more accessible to the lay person, and don’t require urgent attention.

6.0.6 A better website

The website was a necessity, and not the primary focus of this project. Therefore, I prioritised having it work reliably and in a transparent way over sophistication. It needed be compatible with as many web-browsing platforms as possible, which is another reason I prioritised simplicity. It was required at minimum not to impede the progress of the project. I feel it more than met this criteria, but there are areas which warrant improvement going forward. However, the traffic over a longer period of time warrants greater efficiency, and to maintain interest in the site, more user feedback is necessary.

Reloading the entire page every time a comparison is made generates unnecessary network traffic and a higher server load, just to re-transmit the generic template of the page. It would require little effort to alter the page to an AJAX framework which only transmits choice and melody data, rather than the entire site every time. The reason I have not already done this is lack of familiarity with AJAX technologies and the potential difficulties and risks associated with it. In the limited time I had, I chose not to risk introducing vulnerabilities and bugs which I was ill-prepared to diagnose and treat.

Another way network traffic, server load and storage space could be improved would be to serve only the string representation of the melodies rather than the audio. Javascript could then be used to render it into a playable melody. In older browsers and on slower machines, the ability to do this well could be compromised. Also, the objectivity of the experiment may have been compromised, as there is no guarantee that the resulting audio would sound the same on different devices. These are probably not insurmoutable problems, but they were not a priority during this project.

The user experience, as it is, is somewhat basic. I kept it this way deliberately. A few users contacted me to ask if there was any way I could allow them to listen to melodies of their choosing and see the current rankings. I was concerned that users may lose objectivity if they could hear a relatively good melody then had to compare two mediocre ones. I think that was the right decision at the time. Now that the project is over, I have built the listen.php page which allows this, and I might make people aware of it and the rankings. I think having occasional leaks of information during the project would constitute a happy middle ground. As with my interpretations of melody 127, I (or whoever is running any future incarnation of this work) could produce interpretations of the top melody at various points, in order to maintain interest. Provoking rivalry between social media sites could also be an effective way to drive traffic. By recording where people are clicking through to the site from, it would be possible to record contribution levels from different sites and encourage a war to see which site can gain most influence over evolution. This would be combative, rather than collaborative composition!

6.0.7 Interpolation Between Melodies

As a tool for composition, the present system is a good idea generator. For those with a musical education of some sort, it is easy of make small corrections, or mix aspects of separate melodies. For those with limited musical ability, for whom we aim to cater, additional help is needed.

Using edit distance as a measure, we have been able to plot the Euclidean distance between all melodies. This space provides a convenient and intuitive abstraction for potential interpolation between melodies. This representation could be used as an interface to fine-tune favoured melodies. A point in Euclidean melody space between two melody-points A and B would represent a certain number of edits from A towards B, with regular rules about the order in which edits are performed. The rules could be arbitrary, allowing a greater number of melodies to be interpolated.

Chapter 7

Conclusions

I set out to allow people from anywhere in the world to collaborate on music, regardless of skill, experience or proximity to each other. I also aimed to garner information about what people found musical; what makes them feel connected to one set of sounds and not others. In these respects and others the $\mu Tunes$ project was a success: participation in the experiment vastly exceeded my expectations; feedback was positive; every measure of musicality increased far beyond the threshold for noise and the control results; and there were no catastrophes during the project.

It has promise as a composition tool; as my own interpretations and those of Dr. Sidorov prove, it serves as an excellent jumping-off point, or “inspiration generator” for broader works in many genres. Achieving musicality with simple operators and an initial population of what amounts to Gaussian noise is testament to the robustness and versatility of (interactive) genetic algorithms. Musically, this approach constitutes a double-edged sword: on the one hand, our computer-generated melodies were devoid of human biases, predispositions and conditioning, leading to unusual and “imaginative” (if I may say that of a program) outcomes; on the other hand, the process is still relatively blind, and could stand to have knowledge of what is more likely to be good music folded in at almost every stage in order to expedite the process. Besides this, it has shown itself to be effective in the hands of untrained individuals. There is significant merit to this.

There was a reason I chose not to include this knowledge beforehand. I wanted all bias to come from humans, precisely in order to analyse that bias.

Analysis of the evolution of the melodies has shown clearly that there are many factors which humans consider musical and generally like. The analyses relating to the implication/realisation model are of particular interest here, as they specify simple rules by which to generate music that people will probably like. These rules could be captured in future versions of the system. For example, the crossover operator, which currently has no bias, could be biased towards crossing over at points which would realise musical implications in the resulting children.

The cornerstone of this project, the prime mover which influenced the rest of the development, was use of pairwise comparisons in order to gather data. It has proven to be noise-tolerant and strong enough to give us significant, useful data on the human perception of music. Its use was adopted in order to minimise fatigue, but it is difficult to say the extent to which it achieved this as we have no baseline for comparison. Given the large number of participants, it is reasonable to say that it was approachable and not off-putting.

Finally, we have learned that what makes music *music* in the ears of human beings may not be as subjective as we thought. This experiment has shown that we expect melodies to obey rules. It is no secret that formulaic music like pop and folk has an enduring popularity; but it seems that there are indeed firm assertions that can be made about what good music is.

Chapter 8

Reflection on Learning

Every aspect of the project took longer than anticipated. I am reminded of Hofstadter's Law [9]:

> It always takes longer than you expect, even when you take into account Hofstadter's Law.

I attribute this partly to inexperience with a project on this scale, and the fact that I was working alone. A project like this involved the intersection of disciplines which left me spread thin. To see how this compared with my own (admittedly aspirational) predictions, see the appendix, section A.1. Although I have had reservations about group work over the course of my university career, I can see the appeal of working as part of a small, specialised and motivated team.

Despite the workload and deadlines, this has been enjoyable and a good opportunity for me to apply my learning in creative ways. One of the most appealing aspects of this particular project is that it combines two passions of mine: music and science.

A bird in the hand is worth two in the bush. The grass is always greener on the other side. Although I have been aware of these maxims since early childhood, it is not until I had disregarded the tools I was already confident with for an unknown quantity which appeared to offer the world that I truly understood their meanings. In future I will be reticent to employ an unfamiliar tool for an important project.

Chapter 9

References

Bibliography

[1] Humdrum Toolkit Representation Reference melodic interval. Accessed: 2014-04-6.

[2] Php midi class. Accessed: 2014-04-25.

[3] W3Schools html5 audio. Accessed: 2014-04-25.

[4] Weblocks: made with alien technology. Accessed: 2014-04-23.

[5] Meghna Babbar, Barbara Minsker, and Hideyuki Takagi. Interactive genetic algorithm framework for long term groundwater monitoring design. In *Proceedings of the Environmental & Water Resources Institute (EWRI) World Water & Environmental Resources Congress, ASCE, Salt Lake City, UT*, pages 1–10, 2004.

[6] Hector Berlioz. *Grand trait d'instrumentation et d'orchestration modernes, Op.10.* Paris: Schonenberger, n.d.[1844]., Plate S. 996., 1844, enlarged 1855.

[7] Mike Cheng. Lame. `http://lame.sourceforge.net/`, 2014.

[8] Maarten Grachten, Josep-Lluis Arcos, and Ramon Lopez de Mantaras. Melodic similarity: Looking for a good abstraction level. In *Proc. ISMIR*, 2004.

[9] D. R. Hofstadter. *Gödel, Escher, Bach: an eternal golden braid.* Vintage Books, New York, 1980.

[10] Brad Johanson and Ricardo Poli. Gp-music: An interactive genetic programming system for music generation with automated fitness raters. pages 181–186, 1998.

[11] A. Jones K. Sidorov, R. Hawkins and D. Marshall. *μtunes*: A study of musical perception in an evolution context. 2014.

[12] Yaser M. A. Khalifa, Hunter Shi, and Gustavo Abreu. Evolutionary music composer. Submitted to Genetic and Evolutionary Computation Conference 2004.

[13] Ming Li and Ronan Sleep. Melody classification using a similarity metric based on kolmogorov complexity. *Sound and Music Computing*, pages 126–129, 2004.

[14] Xavier Llor, Kumara Sastry, Francesc Alas, and Arquitectura La Salle. Analyzing active interactive genetic algorithms using visual analytics. In *In GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1417–1418. ACM, 2006.

[15] Robert M. MacCallum, Matthias Mauch, Austin Burt, and Armand M. Leroi. Evolution of music by public choice. *Proceedings of the National Academy of Sciences*, 2012.

[16] Bill Manaris, Dallas Vaughan, Christopher Wagner, Juan Romero, and Robert B. Davis. Evolutionary music and the zipf-mandelbrot law: Developing fitness functions for pleasant music. In *In Lecture Notes in Computer Science, Applications of Evolutionary Computing Evoworkshops 2003, LNCS 2611*, pages 522–534. SpringerVerlag, 2003.

[17] Manuel Cebrian Manuel Alfonseca and Alfonso Ortega. A simple genetic algorithm for music generation by means of algorithmic information theory. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3035–3042, September 2007.

[18] Eugene Narmour. *The analysis and cognition of melodic complexity: The implication-realization model.* University of Chicago Press, 1992.

[19] H. Takagi. Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, Sep 2001.

[20] Hideyuki Takagi and Denis Pallez. Paired comparisons-based interactive differential evolution, 2009.

[21] Tuukka Toivonen. Timidity++. `http://timidity.sourceforge.net/`, 2014.

[22] Fabian Wauthier, Michael Jordan, and Nebojsa Jojic. Efficient ranking from pairwise comparisons. In *Proceedings of the 30th International Conference on Machine Learning*, pages 109–117, 2013.

[23] T. Yokoyama, H. Takenouchi, M. Tokumaru, and N. Muranaka. Website design system based on an interactive genetic algorithm using tournament evaluation by multiple people. In *Proceedings of SCIS-ISIS 2012*, pages 2260–2263, 2012.

[24] Tsu yu Wu, Chin te Chen, Kai chu Wu, Ying ping Chen, and Dao yung Fu. Evolutionary interactive music composition. 2006.

Appendix A

Appendices

A.1 Predicted Timescale Versus Reality

My predicted timescale does not bear much resemblance to the course this project actually took. The moderator's comments on my initial report to the effect that it was unrealistic proved absolutely correct

Week 1 •Complete Initial plan.
•Construct basic website for user participation.
•Research evolutionary algorithm techniques
•Research other evolutionary music projects
Reality
Completing the initial plan was the priority, to the exclusion of much else. Brainstormed about the evolutionary algorithm.

Week 2 •Finalise aspects of evolutionary algorithm, eg. representation, crossover/mutation.. •Implement basic algorithm.
•Write code to play string representations as melodies
•Manually run the algorithm to benchmark the number of generations and rough number of step before something musical appears
Reality
Formulated several ideas about how to construct the evolutionary algorithm and ran some tests on them. Did some general research, discovering the general problems of fatigue. Resolved at this point to minimise this effect, as I had no idea of how much participation I would be able

to muster.

Week 3 •Build a back-end for analysis and testing (i.e. Database. It may be possible to incorporate this into the algorithm code)
•Refine and test website
•Refine and test algorithm/back end
Reality
Started using the weblocks framework to try to get a site running. It always seemed like I needed to learn just a little bit more in order to be able to use it properly. I put other aspects of my project on hold to focus on this as without the website, there could be no project.

Week 4 •Launch website
•Publicise and drive traffic

- Canvas social media sites
- Contact fan-base from music career
- Ask other students to participate

Reality
It began to sink in that of the meagre information available on weblocks, little of it even referred to the same version.

Week 5 •Monitor progress (ongoing from now) **Reality**
Finally abandoned weblocks and replaced it with a LAMP stack. Began developing the site which would end up powering the project. From the time spend with weblocks I had at least made a number of design decisions which were transferable, including the design of a logo.

Week 6 •Identify and analyse popular melodies (ongoing)
Reality

Continued developing the website, database and scripts that powered the algorithm. I initially implemented one of my more exotic ranking ideas. I soon replaced this with the algorithm I have today. The site's construction reflects the modular nature of the site and replacing components is straightforward.

Week 7 •Buffer week, allowing time for the unexpected.

Reality

. This is the week in which I first publicised the website and comparisons started happening. That was on the Friday. The rest of the week was spent testing and tweaking.

Week 8 •Analyse final population (this could be in the next week)

•Analyse popular melodies for common features

Reality

I promoted the site more, generally checked to ensure that it was actually functioning as it was meant to under the surprisingly high server load. Some modifications were made to improve performance on mobile.

Week 9 •Write report (ongoing)

Reality

Developed a means to export database contents as CSV files.

Week 10 •Continue writing report.

Reality

Began to analyse results. Created a skeletal report, and began the slow process of filling it in.

Week 11 Hand in final report..

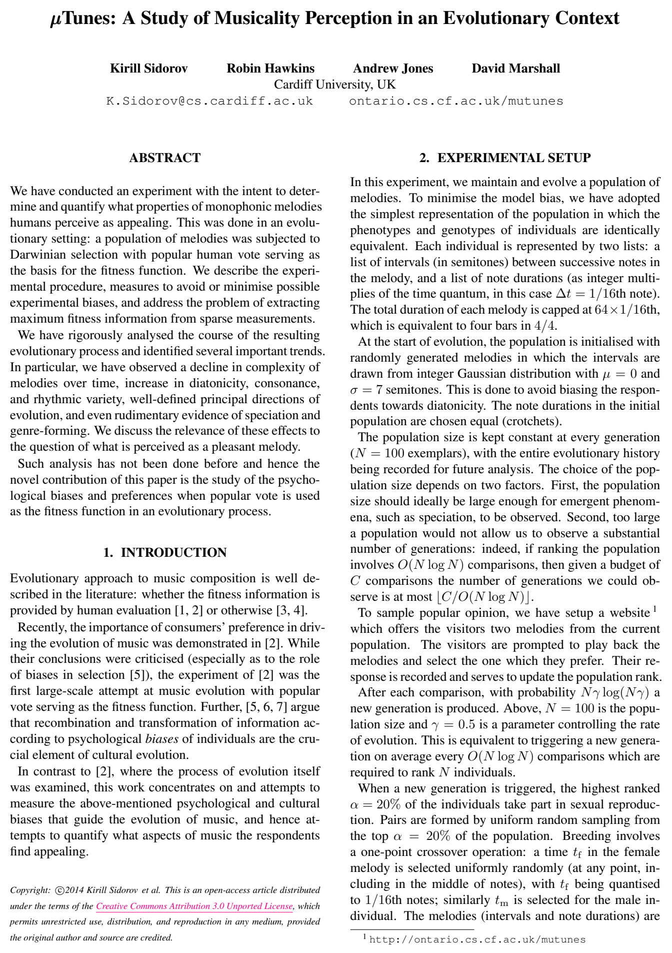
Reality

The Easter break was spent writing the report and analysing the surprisingly abundant data I had collected.

Appendix B

Paper Published

The paper on the findings of my project which I co-authored with Drs. Sidorov, Jones and Marshall. A complete pdf is also attached in the deliverables.



μTunes: A Study of Musicality Perception in an Evolutionary Context

Kirill Sidorov **Robin Hawkins** **Andrew Jones** **David Marshall**
Cardiff University, UK
`K.Sidorov@cs.cardiff.ac.uk` `ontario.cs.cf.ac.uk/mutunes`

ABSTRACT

We have conducted an experiment with the intent to determine and quantify what properties of monophonic melodies humans perceive as appealing. This was done in an evolutionary setting: a population of melodies was subjected to Darwinian selection with popular human vote serving as the basis for the fitness function. We describe the experimental procedure, measures to avoid or minimise possible experimental biases, and address the problem of extracting maximum fitness information from sparse measurements.

We have rigorously analysed the course of the resulting evolutionary process and identified several important trends. In particular, we have observed a decline in complexity of melodies over time, increase in diatonicity, consonance, and rhythmic variety, well-defined principal directions of evolution, and even rudimentary evidence of speciation and genre-forming. We discuss the relevance of these effects to the question of what is perceived as a pleasant melody.

Such analysis has not been done before and hence the novel contribution of this paper is the study of the psychological biases and preferences when popular vote is used as the fitness function in an evolutionary process.

1. INTRODUCTION

Evolutionary approach to music composition is well described in the literature: whether the fitness information is provided by human evaluation [1, 2] or otherwise [3, 4].

Recently, the importance of consumers' preference in driving the evolution of music was demonstrated in [2]. While their conclusions were criticised (especially as to the role of biases in selection [5]), the experiment of [2] was the first large-scale attempt at music evolution with popular vote serving as the fitness function. Further, [5, 6, 7] argue that recombination and transformation of information according to psychological *biases* of individuals are the crucial element of cultural evolution.

In contrast to [2], where the process of evolution itself was examined, this work concentrates on and attempts to measure the above-mentioned psychological and cultural biases that guide the evolution of music, and hence attempts to quantify what aspects of music the respondents find appealing.

2. EXPERIMENTAL SETUP

In this experiment, we maintain and evolve a population of melodies. To minimise the model bias, we have adopted the simplest representation of the population in which the phenotypes and genotypes of individuals are identically equivalent. Each individual is represented by two lists: a list of intervals (in semitones) between successive notes in the melody, and a list of note durations (as integer multiplies of the time quantum, in this case $\Delta t = 1/16$th note). The total duration of each melody is capped at $64 \times 1/16$th, which is equivalent to four bars in $4/4$.

At the start of evolution, the population is initialised with randomly generated melodies in which the intervals are drawn from integer Gaussian distribution with $\mu = 0$ and $\sigma = 7$ semitones. This is done to avoid biasing the respondents towards diatonicity. The note durations in the initial population are chosen equal (crotchets).

The population size is kept constant at every generation ($N = 100$ exemplars), with the entire evolutionary history being recorded for future analysis. The choice of the population size depends on two factors. First, the population size should ideally be large enough for emergent phenomena, such as speciation, to be observed. Second, too large a population would not allow us to observe a substantial number of generations: indeed, if ranking the population involves $O(N \log N)$ comparisons, then given a budget of C comparisons the number of generations we could observe is at most $\lfloor C/O(N \log N) \rfloor$.

To sample popular opinion, we have setup a website [1] which offers the visitors two melodies from the current population. The visitors are prompted to play back the melodies and select the one which they prefer. Their response is recorded and serves to update the population rank.

After each comparison, with probability $N\gamma \log(N\gamma)$ a new generation is produced. Above, $N = 100$ is the population size and $\gamma = 0.5$ is a parameter controlling the rate of evolution. This is equivalent to triggering a new generation on average every $O(N \log N)$ comparisons which are required to rank N individuals.

When a new generation is triggered, the highest ranked $\alpha = 20\%$ of the individuals take part in sexual reproduction. Pairs are formed by uniform random sampling from the top $\alpha = 20\%$ of the population. Breeding involves a one-point crossover operation: a time t_f in the female melody is selected uniformly randomly (at any point, including in the middle of notes), with t_f being quantised to $1/16$th notes; similarly t_m is selected for the male individual. The melodies (intervals and note durations) are

[1] `http://ontario.cs.cf.ac.uk/mutunes`

Figure B.1: Paper on $\mu Tunes$

Appendix C

Deliverables

In the archive submitted with this report, called "deliverables.zip", you will find the following:

C.1 Paper: $\mu Tunes$: A Study of Musicality Perception in an Evolutionary Context

This project is the topic of this paper, hence attaching it. It gives a more concise explanation of the goals and outcomes of this project and is the source of some of the data I used.

C.2 Results

A folder containing two more folders: "csv" and "sql". The former contains 6 CSV files: those containing melodies information for the real results, the control (random) results and those selected for disjunctness; and those containing the comparison information for the real results, the control (random) results and those selected for disjunctness. The latter contains an "sql" file from which you can rebuild the entire mutunes database as of May, 6th 2014.

C.3 Analysis

This folder contains the MATLAB scripts used to analyse the CSV data found in "Results". It also contains graphs and figures generated by these scripts. I have tried to give the scripts names which explain what they do and comment liberally.

C.4 examples of music

A folder containing wav files of popular and influencial melodies from the mutunes project.

C.5 mutunes

All the scripts necessary to set up your own *μTunes*, and a README file explaining the few steps necessary to do so.