# FINGAR, a Genetic Algorithm Approach to Producing Playable Guitar Tablature with Fingering Instructions

Nicholas Trevor Rutherford
Supervisor: Guy Brown

COM3010
May 6, 2009

This report is submitted in partial fulfilment of the requirement for the degree of Bachelor of Science with Honours in Computer Science by Nicholas Trevor Rutherford.

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name:
Signature:
Date:

**Abstract**

This paper investigates systems for converting sheet music to guitar tablature with left-hand fingering data. The guitar fingering problem involves mapping musical notes onto one of several possible fretboard positions and deciding how the hand can form this shape. The problem has been approached as an optimisation problem with several maxima and potentially no most-optimal solution. Genetic algorithms have been shown to be suitable in this problem domain.

A literature review is presented critically examining published work on the problem, some related work in adjacent domains, and highlighting assumptions and common themes seen throughout.

Initial progress is presented in implementing FINGAR, a transcription system using a genetic algorithm to determine optimal playing directions for a piece of guitar music.

# Acknowledgements

Many thanks go to Guy Brown for his valuable feedback and guidance during this project, and Thomas Orbak-Braten for his patience, faith, and proof-reading via emoticons displayed on pasting into a chat window.

Thanks are also due to the developers and maintainers of MacTex, TextMate, Omnigraffle, MATLAB, Java, Skim, the DCS grid engine and anything else which feels left out.

# Contents

# List of Figures

# List of Tables

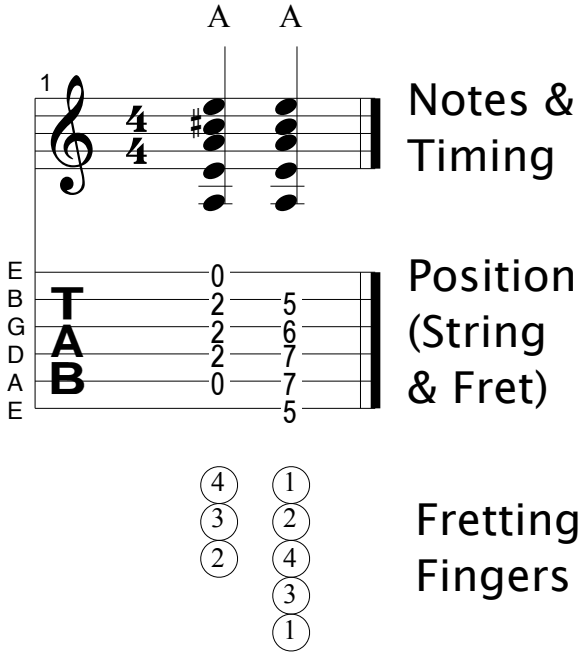
# Chapter 1

# Introduction

Guitar music can be recorded (written) in several forms. Considered here are traditional European sheet music, tablature, and hybrids of the two. These alternative notations contain differing levels of information on timing, fretting, and fingering. Fretting indicates where to play a particular note on the guitar, as each note can be played in many positions. Fingering is the concern of which finger of the left hand is used to play each fretted note. This is usually left open to interpretation, though some arrangements include it as a courtesy.

Figure 1.1 shows the same notes being played in two (of many) different positions. An explanation of the diagram is provided in Section 2.3 and the acoustic significance of alternative positions for a note is considered in 2.4.2. Different positions directly affect a piece in terms of attainable speed, control, and the difficulty of performance. We consider an optimal arrangement to be one which is comfortable and sufficiently easy for the musician to play, though this may be expanded on to consider styles and particular techniques.

Figure 1.1: Two A chord positions




The aim of this project is to develop an algorithm for producing fretting positions and Left-hand fingering data (LHFD) for arbitrary pieces of guitar music. Suitable input data could be produced automatically by codified interfaces or adapters for any digital encoding of traditional sheet music, i.e. formats storing instructions for playing music rather than a recording of it being played. Examples of suitable formats include MusicXML, MIDI[1], GuitarPro[2] and other formats where musical note and timing information may be extracted. It is also possible to deduce this information from digital tablature where timing information is provided.

Given a score to process fretting and fingering arrangements will be deduced and assessed in terms of their perceivable quality and acceptance by human players.

The literature review presents a survey of published works on this subject, comparing approaches and their respective merits and failures. It is found that the problem is not solved,

[1] Musical Instrument Digital Interface

[2] A popular tablature input application with a proprietary file format. A specification for GP4 files is available freely online at http://dguitar.sourceforge.net/GP4format.html

that an optimisation approach is most suitable and that there is considerable room for further work on divergent approaches. Assumptions and simplifications made by different treatments are identified and discussed in terms of information loss and computational tractability. It is also found that the subject has no standard model to work from and that no study has been performed judging the human acceptance or utility of such a system.

Following is a discussion of the design and implementation process of FINGAR, a genetic algorithm solution to the left-hand fingering problem for monophonic pieces. Experimental results and a qualitative evaluation of its performance are provided. Practical issues encountered and further work are also presented.

# Chapter 2

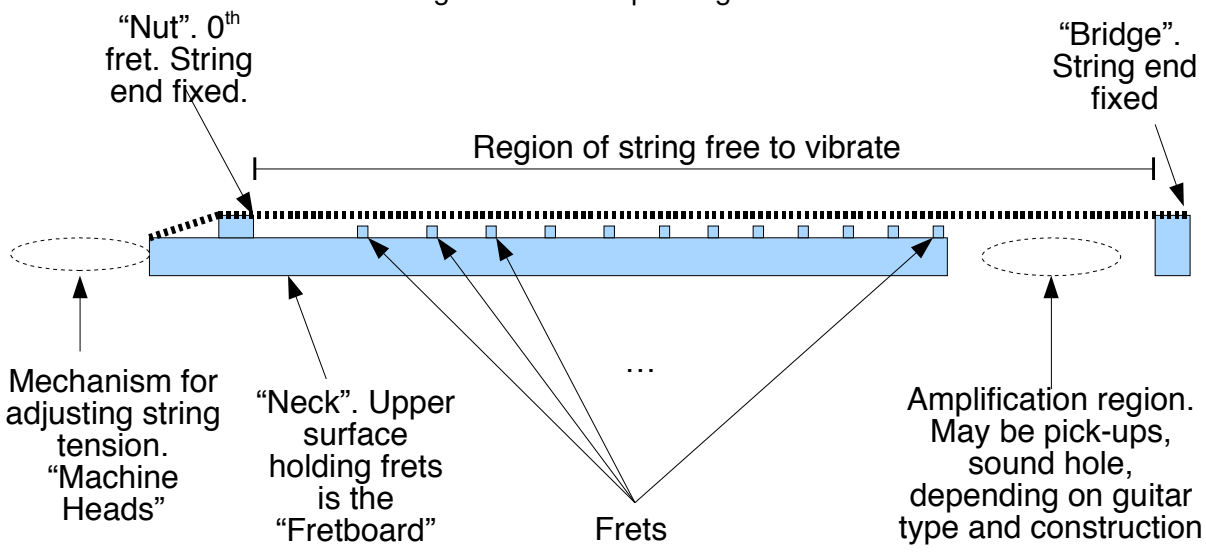
# An Introduction to the Guitar, for Computer Scientists

## 2.1 The Basics

The guitar is a musical instrument which produces sound by amplifying the vibration of its strings. The strings sound when plucked, struck, or otherwise caused to vibrate. They are attached to the body of the guitar at either end, their length and tension determines their fundamental frequency when played as an open string (i.e. not fretted with the left hand). The strings are tuned to standard frequencies with an electronic guitar tuner or equivalent.

We shall consider an abstract guitar[1] to obviate details of construction which are not relevant[2] to the fingering problem. The portions of interest are the guitar strings, the fretboard, and the guitarist's left hand. It is also important to understand how to interpret the tablature diagrams presented within, so this notation is also explained.

Figure 2.1: A simplified guitar




[1] The model is derived from an electric guitar, hence the high number of playable frets.

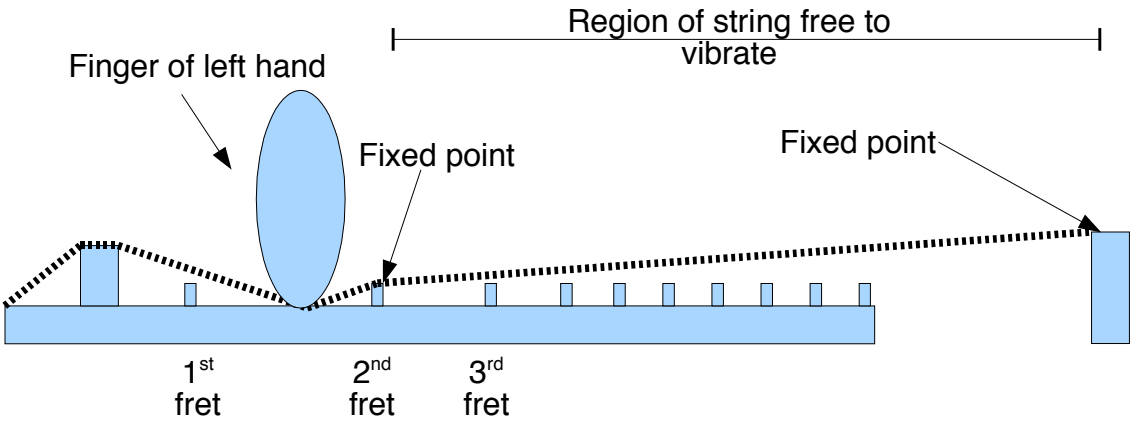
[2] alternatively this may be considered an abstraction used to simplify the problem and produce a general result. [HM02] considers the affect of the guitar body on fingering decisions

By holding the string against the fretboard with the left hand[3] the length of the string allowed to vibrate is reduced and the fundamental frequency of vibration increased. This length difference can be seen by comparing Fig 2.1 with Fig 2.2.

The fretboard is a wooden board with carefully positioned metal frets embedded on it. The purpose of these frets is to "discretise" the lengths that the string can be shortened to, and hence the frequencies it produces. This means that as long as two guitars have been produced correctly, and are in tune, they will produce notes at the same frequencies when played in the same way.

That is not to say that the guitar cannot play other notes, techniques such as string bending allow a guitarist to take one of these standard note frequencies and increase it in a continuous manner by exerting a lateral force on the string to stretch it further, in addition to holding it against the fretboard.

Figure 2.2: A guitar being played at the 2nd fret




The guitarist's hands are obviously important in playing the instrument, but for this work only the left hand is considered. In simple playing the right hand is used to sound notes, by plucking or striking, while the left hand is used to manipulate the strings against the fretboard, and so control the vibrational frequency of the strings, and thus the notes being produced. This work is interested in solutions to the optimal positioning of the left hand while playing a piece. The right hand is considered a different issue, and perhaps a less interesting one as there is less obviously encodable variation.

For details of the assumptions made about the guitarist and guitar in this work see Figure 2.3.

## 2.2 Basic Music Theory

Taylor [Tay91] provides an informal introduction to musical notes and the concept of an octave based on the piano keyboard. This is presented in a truncated form here.

A piano is tuned to produce particular frequencies when certain keys are pressed. These frequencies are predetermined, the same is true for many other instruments, including the guitar. The white keys on the piano are grouped into 8s called octaves, where the $1^{st}$ and $8^{th}$ key in succession are the same note in a different octave. This can be seen in Figure 2.4.

[3] ignoring advanced techniques such as right-hand tapping

Figure 2.3: Guitar Assumptions

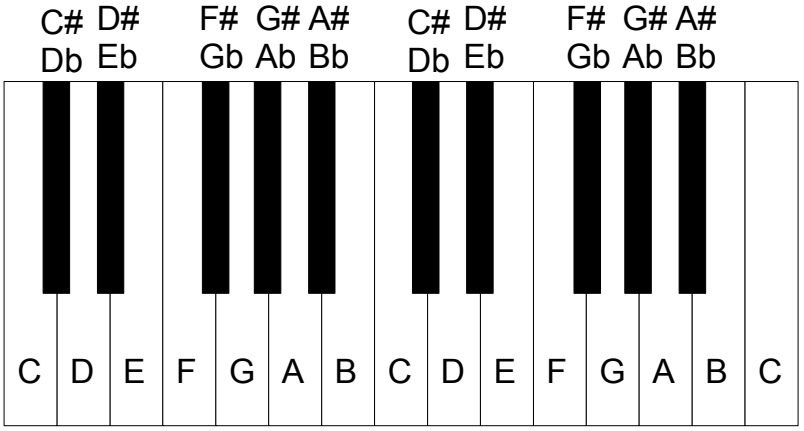
A typical guitar is considered:

- to have 6 strings.
- to use the standard EADGBE tuning
- to be played by someone with the full use of all 4 fingers on their left hand.
- to have 24 playable frets in addition to the open string (0th fret)

Notes are cyclic, pick any note and advance by 8 white keys and you will play all of the named notes, but in a different sequence.

The black keys are sharps and flats, which can be thought of simply as 'in-between' notes. For this paper they should be considered distinct notes, entirely divorced from their namesake except for proximity. No knowledge is assumed or use made of musical theory pertaining to the relationships between notes. It is enough to consider them as an enumerated set of discrete frequencies.

Figure 2.4: An example piano keyboard




The chromatic scale is all of the keys on the piano keyboard, black and white. The gap between a white key and its adjacent black (or white, if there is no black) key is called a semitone interval. An octave contains 14 semitones, travelling from the starting note to itself in a higher octave.

Figure 2.5 shows the spread of notes across the positions of a guitar. The number associated with each note in the diagram states the octave which the note belongs to, where E is used as the root note.

Figure 2.5: A 14 fret guitar's note and octave positions

| Fret | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| String E | E1 | F1 | F♯1 | G1 | A♭1 | A1 | B♭1 | B1 | C1 | C♯1 | D1 | E♭1 | E2 | F2 | F♯2 |
| String A | A1 | B♭1 | B1 | C1 | C♯1 | D1 | E♭1 | E2 | F2 | F♯2 | G2 | A♭2 | A2 | B♭2 | B2 |
| String D | D1 | E♭1 | E2 | F2 | F♯2 | G2 | A♭2 | A2 | B♭2 | B2 | C2 | C♯2 | D2 | E♭2 | E3 |
| String G | G2 | A♭2 | A2 | B♭2 | B2 | C2 | C♯2 | D2 | E♭2 | E3 | F3 | F♯3 | G3 | A♭3 | A3 |
| String B | B2 | C2 | C♯2 | D2 | E♭2 | E3 | F3 | F♯3 | G3 | A♭3 | A3 | B♭3 | B3 | C3 | C♯3 |
| String E | E3 | F3 | F♯3 | G3 | A♭3 | A3 | B♭3 | B3 | C3 | C♯3 | D3 | E♭3 | E4 | F4 | F♯4 |

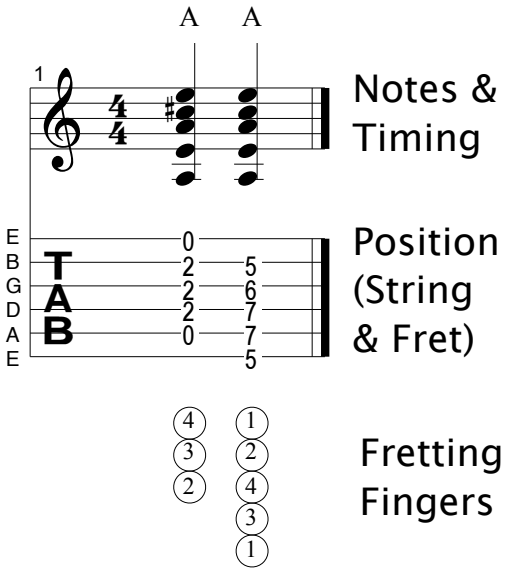
## 2.3 Tablature Diagrams

This section aims to explain how to read the LHFD augmented tablature diagrams present in this paper. Figure 2.6 shows two positions where an A chord can be played. The upper stave is

traditional sheet music. The lower stave is tablature. Left hand fingering data (LHFD) is shown below the stave.

Tablature is read as the top line of the diagram being the top string of the guitar (with the highest pitch, actually found at the bottom, closest to your leg) and each string being represented as a line on the tablature. A guitar fingering position consists of 3 parts: string, fret, finger. Knowing the line on the tablature gives you the string. The number written is the fret. The finger is often omitted, or noted in numerous ways. In this paper the GuitarPro notation is adopted as it was used to produce the diagrams. It is fairly intuitive, and displays just enough information to be useable.

Figure 2.6: Two alternative A chord positions and fingerings




To play the first A chord you would position your left hand's little finger (4) at fret 2 of the B string, ring finger (3) at fret 2 of the G string, and middle finger (2) at fret 2 of the D string. The top E string and A string are played at fret 0, which means they are played 'open' without any fingers of the left hand pressing the string down. The right hand then plucks or strikes all the strings except the bottom E string, which is not indicated to be played in the diagram.

To play the second A chord you move your left hand so that your index finger (1) forms a 'barre' chord across the 5th fret, holding down all the strings at once. The remaining fingers are positioned to the right of it, shortening the strings further. In this case the top E string is not played.

The horizontal axis represents time progression (though it is not measured, that is represented properly by traditional sheet music).

## 2.4 Domain Space Complexity

### 2.4.1 Note positions

Given the range of the instrument as 4 octaves, this makes the total number of notes[4]:

$$\textit{notes in range} = \textit{octaves} \times |\textit{octave}| = 4 \times 12 = 48 \tag{2.1}$$

From the number of strings and frets we can find the number of distinct playable positions:

$$\textit{total positions} = \textit{number of strings} \times (\textit{number of frets} + \textit{open string}) = 6 \times 25 = 150 \tag{2.2}$$

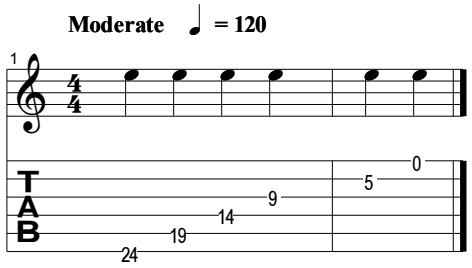
Each of these positions represents a single note in the range, where a many-to-one relationship exists between the positions and the note as seen in Figure 2.5. The distribution of notes is not uniform: at the range's extremities notes can be played in only one position, where as towards the middle of the range there are at most 4 positions for an individual note.

This can be proved by considering that the relative spacing between strings is 5 frets, except for G to B which is 4 frets. That is to say if you play fret 5 on the bottom E string the same pitch note is produced as when playing an open A string. The positions (Low E, 10), (A, 5), (D, 0) all produce the same pitch D note in the instrument's first octave.

[4]Where sharps and flats are considered distinct notes

The maximum positions for a note can be discerned by working in from the upper bound based on the rules of the physical system. Each position on a string plays a different note. There are 6 strings on the guitar. The highest note on the lowest pitch string will show whether it is possible to to play a note at 6 positions.

Figure 2.7: E 4th octave positions




Although showing that 6 positions for a note is possible (Fig 2.7), it also demonstrates that this is an upper limit, and only possible on a guitar with 24 playable frets[5]. Note that on the top E string the 0th fret is played, i.e. an open string. It is not possible to play a lower pitch note on this string. It is also not possible to play a higher pitch note on the bottom E string, as 24 is the highest fret. This implicates that increasing or reducing the note by 1 degree will result in one or the other string not being able to play it, and thus 5 rather than 6 positions existing. The important information to draw from this is that the maximum space occupied would be

$$\textit{upper bound} = 6 \times \textit{notes in range} \tag{2.3}$$

which is a computationally tractable amount, and that the actual space occupied is lower.

This demonstrates only that the domain can be represented, not that the problem space can be searched effectively. Heijink and Meulenbroek (HM) [HM02] state that for an $n$ note melody played entirely on a single string the number of fingerings is $4^n$. Tuohy and Potter (TP) [TP04] instead consider positions, stating a search space of $3^n$. Combining these search spaces to consider both fingering and positions will clearly lead to one that is greater still, $(3 \times 4)^n$ for monophony. A higher value of $16^n$ is stated in [DPRL04]. This difference is due to considering polyphony where several notes are played together and progressively fewer fingers available, but the difference is not important. For the lower order it can still be shown that the search space quickly becomes computationally intractable when considering real length pieces of music. For example the Sailor's Hornpipe (traditional) used during testing was approximately 60 notes in length, which for HM's limit gives a search space of the order $10^{28}$, this being an artificially restricted lower bound. A value for positions and LHFD would be much higher, $10^{64}$. Internet sources suggest $10^{80}$ atoms in the universe, $10^{49}$ making up planet Earth, and $10^{50}$ for the Chess game's search space. Clearly the search space is intractable with regards a complete search.

Large search spaces are not a new issue, being encountered often in artificial intelligence search problems. Approaches applied to this problem range from function approximation by machine learning, to stochastic methods, to domain simplifications such as independence assumptions and framing similar to those seen in Automatic Speech Recognition (ASR). These are explored in the literature review, Chapter 3.

### 2.4.2 Musical Significance of Multiple Positions per Note

Academic opinion over the importance of fingering on the sound produced is split, some opting to consider string length, harmonics and finger-fret damping as important factors [ST88, RD04], while others ignored the issue or stated it outside the scope of their work [MY02, TP04]. Indeed

[5]Many guitars have fewer frets, indeed nylon strung classical guitars would typically have closer to 12 playable frets.

this was the subject of a study by Traube and Smith III in [TS00], where they investigated the discovery of “plucking point” and “fretting point” from audio recordings with some success. HM’s behavioural study [HM02] focussed on biomechanical concerns but commented that tonal and cognitive considerations were also pertinent.

For the purposes of this paper all positions for a note are considered equally valid, the selection of notes based on how they sound is seen as an arrangement issue best left to the performer; by providing them with all found alternatives they will be well equipped to select or derive their own favourite solution.

## 2.5 Advanced Techniques

The “hands-on” manner in which the guitar is played allows a guitarist to employ a wide variety of physical techniques to change the sound produced by the instrument, not considering the plethora of electronic methods available. These range from changes to how the strings are depressed by the fingers, e.g. modifying the applied pressure, to more novel techniques such as using a knife or bottle to depress the strings.

Presented here is an introduction to some of those thought relevant to the left hand fingering problem, and those mentioned in research papers in the literature review (Chapter 3).

### 2.5.1 String Bending

When a guitarist is playing a fretted note they hold a string against the fretboard behind a particular fret. This then produces a particular note determined by the fret, as previously described. The guitarist is able to change the tension in the string by lateral movement. This increased distance on both sides of the “pinch point” (which can be seen as an application of Pythagoras’ triangle) causes the tension in the string to increase and raises the pitch. The relationship between the increase in distance from fret to bridge and the increase in tension is outside the scope of this paper, but the effect is well known and commonly used to increase the note pitch.

The musical effect of this is a legato transition between notes. The tension in the string may require that stronger fingers, or multiple fingers, are used to bend the string, which will affect the guitarist’s choice of fingering pattern in the preceding and following notes.

### 2.5.2 Glissando Slides

Glissando slides are enacted by holding a fretting position and moving the finger up or down to the next desired position without losing contact with the fretboard, this causes the pitch to change in a continuous (though stepped, because of the frets) manner.

A change of left-hand position (LHP) is often, though not always, implicit in a glissando slide.

### 2.5.3 Hammer-on and Pull-off

These techniques add energy to the string vibration with the left hand during note transitions. A hammer-on is simply putting a fretting finger into position harder than normal while the string is vibrating, the effect is similar to the internal workings of a piano.

A pull-off is more complex, but not dissimilar from string bending. To perform a pull-off the string is bent a little then released suddenly, mimicking the effect of plucking a string with the right hand, and the sudden release adding to the string's vibration energy.

Both of these techniques introduce legato slurring into a sequence of notes. In traditional sheet music the effect would be represented by an articulation mark called a slur. This is typically written into the music, as it changes how the music sounds.

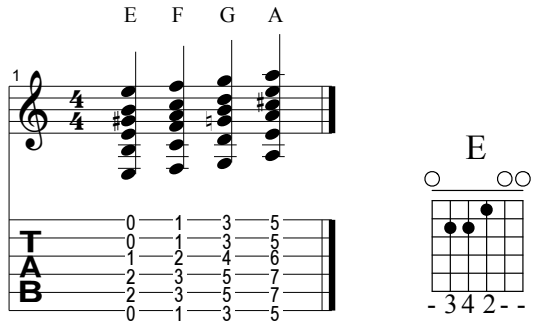
These techniques are often added to pieces according to personal style. A player adding them in this way could be interpreted as implying additional grace notes.

### 2.5.4 Barre Chords

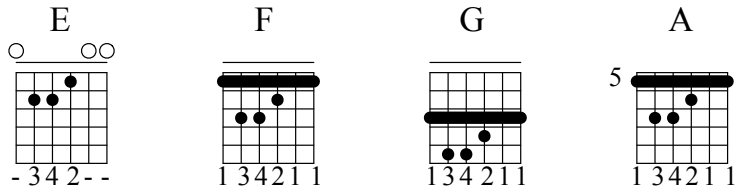
A barre chord describes a chord shape where the index finger is used to fret several strings at once, and the remaining fingers are used to play as normal. This allows common chord shapes used in the open string positions to be moved higher up the neck to make other chords. For example Fig 2.8 shows the E chord being used to play E, F, G and A chords.

This technique is widely used but can be difficult for beginners, as it requires considerable lateral force to be applied across all of the strings to fret the note properly, without damped or buzzing sounds when played. Adjacent strings are also often fretted in a similar fashion with other fingers.

Figure 2.8: E Barre Chord Examples




(a) E Barre Chord Notes and Positions




(b) E Barre Chord Shapes

# Chapter 3

# Literature review

This section presents three main published approaches to the guitar's left-hand fingering problem. The papers presented evolve from Sayegh's seminal work on applying the optimum path paradigm to the stringed instrument fingering problem [Say89]. Sayegh presents two approaches: an expert system approach, and an optimisation approach. Informally it could be said that an expert system typicaly considers and produces a single solution according to its inbuilt logic, while the optimisation approaches (presented in Section 3.2) consider many alternatives and find the best according to some criteria. Presented here are papers from both of these categories, and different approaches to the optimisation approach in particular.

Readers unfamiliar with any of the musical terms used here may find Chapter 2, and Appendices C and B of use, which provide descriptions of the musical fundamentals and physical guitar playing actions mentioned in this paper.

## 3.1 Expert systems

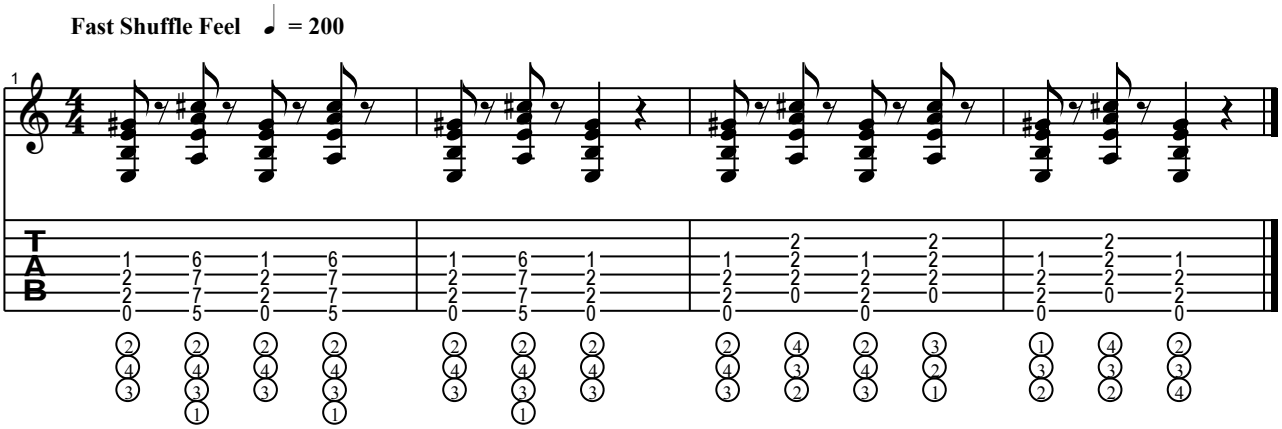
Expert systems approach the guitar fingering problem in a logic driven manner, taking the presented problem and applying rules to the provided data until a result is determined. These rules mimic a human expert. Examples of expert systems for the guitar fingering problem have been published by Sayegh [Say89], Miura and Yanagida (MY) [MY02], and Emura et al [EMHY06].

MY's initial rule-based system considers only monophonic pieces, but is still of interest as a baseline. It uses "Phrasal segmentation" to split the piece into phrases (sections) played with a "static" hand position. This means that within a phrase the fingers may move to different strings and frets within a particular region of the neck, but because the hand will not move up or down the neck as a whole each phrase will be constrained to a region 4 to 6 frets wide.

These phrases are determined in a simple automated manner: notes and rests (pauses) longer than predetermined time constants are found and used as phrase boundaries. A starting position is selected by the user, then these phrases are allocated left-hand positions which minimise the overall hand movement. It then processes each phrase in turn, allocating the appropriate fingers to frets. As a post-processing stage they add embellishments to the piece where the notes match some filtering rules.

MY's algorithm runs across a reduced hypothesis space, accomplished by asking the user to select an initial position. The algorithm has no concept of static cost: it prefers tablatures with a minimum of hand movement. This simplifies their solution, but introduces an issue of local maxima to their solutions and doesn't deal with difficult static shapes or excessive finger movement within a phrase. A more terminal concern is that their program halts when faced

Figure 3.1: Minimum Hand Movement Counter-example
Bars 1-2 are the suggested barre-chord solution omitted by MY. Bars 3-4 are the least hand movement solution, though the LHFD displays alternatives rather than being optimally playable.




with a phrase requiring more than 7 frets. This may well occur, and rather than halting should prompt some backtracking to their phrase determination algorithm, or some other alternative process. This issue of dead-ends is not unfamiliar to issues in tree-searching, and gives an example of an expert system failing where an optimisation approach would inherently succeed.

Another problem with this approach is it does not recognise a minimisation of finger movement as beneficial; it seems sensible to assume that a single motor action moving the entire hand is less complex than controlling the movement of several fingers at once to different locations. Indeed this is suggested by HM's behavioural study where 3 fundamental criteria are identified: hand movement, finger movement and finger span.

Putting aside the cognitive aspects, which are beyond the scope of this paper, this can be shown empirically. It can be demonstrated by the rhythm guitar phrase in Figure 3.1 that the solution with the least hand movement is not necessarily the best solution in terms of difficulty, control, tonal quality or speed. A particular technical advantage of the barre chord solution is that it becomes trivial, part of the natural hand movement even, to dampen the strings when moving the hand, thus making the notes end when they should instead of ringing on and overlapping with the next chord as they do with the open string chord solution. In some cases this would be desirable, in others it would not, according to the desired playing style.

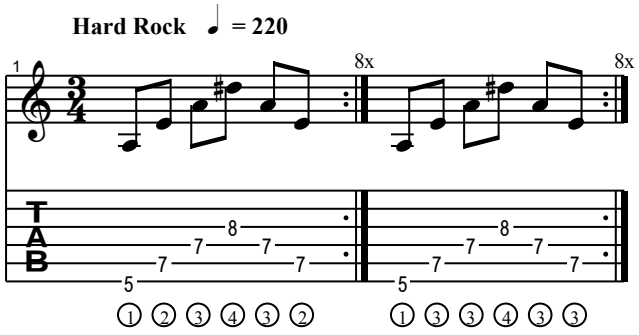
Although contrived, this example is not beyond a novice guitarist. It highlights the dichotomy between finger movement and hand movement, providing an example of this flaw in MY's model. While the shown example is polyphonic it could be played as a monophonic broken chord.

MY's approach to finger allocation (LHFD) is that within a phrase, which has already been allocated a left-hand position, fingers are allocated in the same way as they are when playing scales [Win02]. This is a safe bet for monophonic melodies, particularly for beginners who cannot play barre chords, but is not optimal in all cases. Figure 3.2 is an example high-speed monophonic melody based on a chord shape. It is played without any left-hand finger movement. The notes on the 7th fret could, for example, be played using the 2nd and 3rd fingers, or the 3rd finger depressed across both strings as a barre chord. Which of these is preferred depends on a number of factors including the mechanical difficulty of the alternatives (barres can make some chord shapes more comfortable) and what comes before and next in the piece. When LHFD is being determined barre chords should be considered in addition to standard movement, and preferably indicated in the output. Standard chord diagrams may provide an

interesting approach to this.

MY's work considers glissando slides, hammer-ons, pull-offs and whole or half tone string bending (which they refer to as "chalking-up"). Their interpretation of this issue is to let the algorithm decide where to add these techniques to music, rather than applying them as instructed by tablature and using them in their cost functions. Their work here is not really in the spirit of the problem it is attempting to address; rather than fulfilling the specified piece of music they are modifying it by inserting embellishments themselves regardless of musical merit. For a consideration of automated guitar music arrangement, which is effectively what this leads in to, see [TP06a]. One possible improvement would be to expand their idea of "Phrasal segmentation" to take glissando slides into account, as these require a change of hand position and sound quite different to the same fretting change without a slide.

Figure 3.2: The arpeggio sweep problem




The evaluation used by MY's is a fret-span comparison with available software and a human transcriber, where the solution with the smaller fret-span is considered better. They obtain good results, but do not address the issue of whether their tab is optimally playable, or suited for a beginner as they intend. A survey of the produced tablatures would have been helpful in assessing this.

It should also be noted that no comparison of musical equality is performed, for example whether their algorithm's arrangement is as true to the original piece as a human transcription, this would be particularly important when their embellishment options were turned on, to introduce glissando slides, hammer-ons, etc.

A final concern with MY's work is that they do not cite or offer a comparison with any previous research in the area.

MY later contribute on Emura et al's [EMHY06] chord sequence system. This is a quite different system. It is preprogrammed with a fixed collection of chords and given any sequence of these chords it will produce an optimal sequence of alternative fingerings, according to its rules on what constitutes an easy chord shape. Alternative chord fingerings are produced from stored knowledge of chord composition, this space is then pruned according to some heuristics.

Unfortunately parts of the paper are patchy, the pruning comparison omits results on "Chords and fingering" claiming that "fingering is not specified in chord-books"; although a reference is not forthcoming this author recalls several books providing this information for chords. The system is unable to produce results for chords outside of its knowledge. Again no reference is made to other academic works. Despite these shortcomings it makes use of an intriguing evaluation method using a MIDI guitar to record several users sight-reading the chord sequences produced by their system and using chord book references, then performing a statistical error analysis between the recorded notes played and those instructed to play. Since this was carried out for several musicians of varying ability this approach gives an indication of how difficult the solutions from each approach were to perform. It is not made clear how sequences were constructed from the chord positions shown in the reference books, whether impartial, skilled, or otherwise.

Both of these papers provide details of cost or viability heuristics, one for monophonic play

and one for chords, which may form helpful starting points for optimisation system cost functions, as follows in 3.2. Indeed the cost functions for FINGAR described in 5.3 were inspired by MY's notion of left hand position.

## 3.2 Optimisation systems

An optimisation problem is one where multiple competing solutions are ranked by efficacy to determine which is the best. This approach is common in machine learning, and there are various well established techniques available. The variant in this approach is the evaluation function used to determine how good a solution is, this function is the general description of the cost and fitness functions mentioned herein.

A useful application of optimisation is in least-cost path discovery. Given a graph of states and transitions between them a path is found through the states which accumulates cost as it progresses. The optimisation problem here is in finding the cheapest path from start to finish. An example of this is estimating the fastest way to travel from one train station to another, where there may be a single direct train or it may be necessary to consider several different routes and changes of train. It should be noted that the concepts of start and finish, and allowable paths, will vary by context and the type of graph being used. For an introduction to graph theory and algorithms see [Ger07], or similar core texts.

The cost function is crucial to an algorithm's efficacy: it makes the decisions about which features or stochastic patterns are of interest and how they affect the cost of a path. If the cost function is unreliable then the search algorithm will produce incorrect results through no fault of its own.

An important issue in graph searching is the search space size, with implications for memory and processing time. A graph may be too large to hold in memory in its entirety at one time, for example the games Chess and Go have very large search spaces (as mentioned in Section 2.4). This issue is more difficult when considering time, performing a complete search of $10^{80}$ possible paths at a rate of 1000 paths per second would take $3.17 \times 10^{69}$ years. Given that the age of the universe is believed[1] to be $1.37 \times 10^{10}$ years it is clear that with current technology this is not a tractable search problem.

This is an old problem and has been worked on by great minds since the early days of computing. Dijkstra's algorithm is an optimal general graph search algorithm. Viterbi's algorithm is a version of this simplified by assuming layering in the graph. Other search algorithm's of note are A*, Bellman-Ford, and many others which may be explored in numerous texts and Wikipedia's section on graph search algorithms[2].

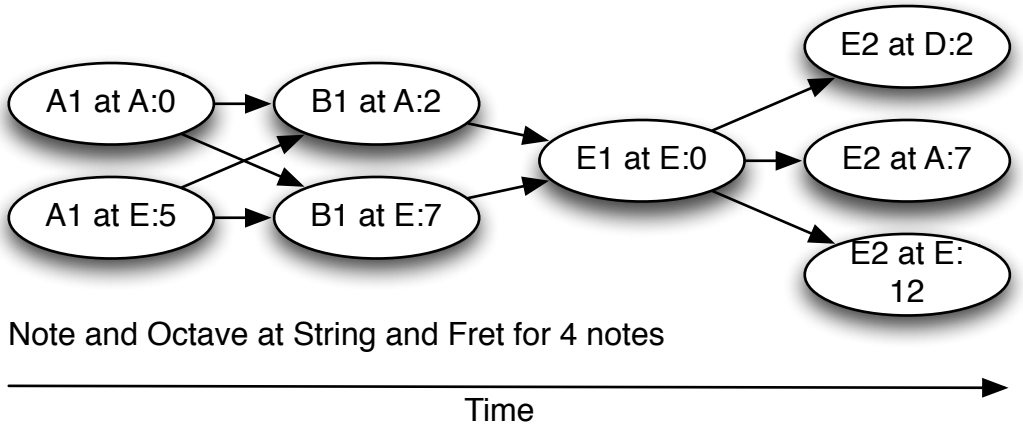
The important concept to capture is that the search will retain or lose certain properties when diverging from a naive complete search. Dynamic programming approaches store calculated values, then when a point is revisited its cost is looked up rather than calculating the entire path again. This optimises the search time without loss of results. Other approaches may decide to follow the most obvious solution they find (hill-climbers), the simplest (breadth-first), or pursue a certain possibility and ignore all others (depth-search). These ideas may combine into several different algorithms as previously discussed. Issues arise as a result of these optimisations: some algorithms will find a solution, but it might not be the best solution. Even worse, some will not find a solution when one exists.

Stochastic methods are another approach which randomly search points of the search space for good solutions and have some criteria for stopping their search, perhaps as soon

[1] http://en.wikipedia.org/wiki/Age_of_the_universe

[2] http://en.wikipedia.org/wiki/Graph_traversal

Figure 3.3: Hypothetical Fingering Solution Space




as they find a solution, or one satisfying certain heuristics, a 'good-enough' solution. Although random these searches may be directed by their results in some way, which leads on to the idea of a Genetic Algorithm introduced in 3.2.3.

### 3.2.1 State representation of guitar fingering

In the guitar fingering domain the act of playing a piece of music can be represented as a path traversing fingering states (Figure 3.3), i.e. a sequences of edges across the vertices of a Directed acyclic graph (DAG). This path crosses the graph one note at a time, where each note has several possible fingering states in parallel and each path traverses one of these states. The path traversal accumulates cost as it progresses from start to finish, from either the states, transitions, or both. This results in many alternative paths with different costs, and the optimisation process attempts to find the cheapest path (though may settle for a lesser one, depending on search technique). The notion of completeness does not apply in this problem, as all paths through the graph are solutions to how to play the piece. Admissibility is an important factor since this is an optimisation problem, but not essential if there are competing maxima. A solution found quickly which is good enough to play, and can be tinkered with by the guitarist, may be preferable to the best solution if it takes several hours or longer to find, if indeed it can be determined to be the best at all without a complete search.

Clearly this results in a large search space. This encoding expands to polyphony for chords where constituent notes are of homogeneous duration, but breaks down when considering interleaved asynchronous melodies. The state notation demonstrated thus far cannot apply as the states overlap in the time domain, a more complex notion of progression through a piece and how notes begin and end is needed. A possible intermediary approach is suggested by Huang and Panne (HP) and found in Section 3.3. The presented works were not forthcoming with details of their encoding policies; it will be necessary to devise one as part of any attempted work. The encoding is important as it will determine the information available to any evaluation function.

The rest of this chapter considers the two published approaches to this optimisation problem: Dynamic programming (DP) and Genetic algorithms (GAs).

### 3.2.2 Dynamic Programming

Dynamic Programming, developed by Richard Bellman in the 1940s [Dre02], is a notion of breaking a large problem up into smaller problems, and reusing the solution of simple problems to build up solutions to larger problems. The advantage of this is that where two or more solutions make use of the same smaller pieces these smaller pieces need only be calculated once and the result shared. An analogy from another domain could be the sharing of objects in an object oriented programming language, where object references are passed into functions rather than passing a distinct copy of the object and all its values.

The earliest work cited is by Sayegh [Say89, ST88], who discusses an expert system approach implemented with Prolog from an earlier work[3], and at length treats the problem as a machine learning exercise using the Viterbi algorithm to find the optimum path, where an evaluation function has been produced with reinforcement learning.

The Viterbi algorithm is an interesting choice, as if tractable and correctly costed it is guaranteed to find the optimal solution. It is an algorithm favoured in speech processing as an efficient way to find maximum likelihood results for Hidden Markov Models [GM00, DJ08]. The use of such a search algorithm shifts the focus of the work from the search task, which can be assumed solved if correctly encoded and computationally tractable, to the cost function training problem.

An important flaw with this model is the claim made by Sayegh in [ST88] that "The individual costs depend on the two fingerings under consideration and are not directly affected by further neighbouring fingers. This assumption is both reasonable and corroborated by experimentation", this will be referred to as the *Adjacent Note Assumption*. This is a necessary assumption of the Viterbi algorithm which is suspect in complex domains. While a valid assumption for low level guitar playing, such as a short sequence of polyphonic notes or scales, it does not scale up well to more complex pieces involving cognitive considerations such as minimising the different positions to remember for a piece, reoccurring chords and subtly different note patterns. For these to be considered by a cost function it *must* have the ability to look back further than one note. Indeed this is criticised by TP [TP06b] in their justification of a genetic search for this problem.

It is important to state that by saying the model is flawed does not mean it will not work, rather that it is imperfect. Assumptions such as these see use in systems for ASR, biological modelling, and many other scientific and engineering ventures. This may result in a divergence in future research, analogous to works using linearity assumptions having diverged from those studying non-linear methods in other fields. These assumptions are in place for good reason: they make the problems more tractable. In this case they make the search problem soluble with a complete search at the cost of some information about the problem. That is, the system sacrifices admissibility for some cases to simplify the search overall.

Moving chord shapes present a possible issue with the Adjacent Note Assumption in polyphony. A chord shape being moved up the fretboard, all fingers remaining in contact with the same strings but sliding up the frets together, and only one string being played. This single note will pay the transition cost from the previous position. If some of the other strings are now played it is not clear what will happen: do they pay the transition cost from the previous position they were played at, or pay no cost at all? Either approach has issues, the first results in erroneously paying for transitions more than once, and the latter would not assign a cost to a potentially difficult finger position change in "mid-air" which will be missed by the transition cost because the finger was not in use in the previous note. An example of cases where the latter will let down the system is playing a barre chord then moving it up the neck to another position

[3] which unfortunately was not available to review

and playing the same chord (so just sliding the whole hand up, not moving fingers). This is easy to do, and is the basis for much simple rhythm guitar music. If the notes in the chord are staggered in time (i.e. a broken chord or arpeggio) rather than played all at once they will be treated as going in and out of use, with an associated cost, despite not having moved. In this sense it is preferable to have either an indication of the state of all fingers positions, even those out of use, or an explicit encoding of the left hand's position, as used in [MY02].

Radisavljevic and Driessen (RD) [RD04] implement and build on Sayegh's idea of DP (with an algorithm akin to Viterbi's). Their interest is in tuning their algorithm to particular styles of music through training over selected scores. RD's treatment of the domain omits glissando slides and barre chords amongst other techniques, but it tackles the two hard problems: left hand finger data and polyphonic music.

A two-part cost function is used in determining the optimal path, a static cost function to judge the difficulty of static positions and a transition cost function, which judges the difficulty of movement between states. These cost functions are feature-based (or heuristic), justified as avoiding intractably large look-up tables that result from the combinatorial explosion of position transitions. This is akin to the other works presented.

Their algorithm is tuned for a particular style through "Path Difference" learning. This optimises their feature weight coefficients, training their algorithm with human tablature of the desired style. This technique is analogous to Hebbian learning and training stages in other machine learning paradigms, such as those presented by [Mit97]. They use a gradient descent update technique to reduce the path difference between the training data and the tablature produced by their DP algorithm.

By using a DP approach and these two costs the search space implicitly captures hand movement (LHP) by preferring paths with lower cost transitions unless they involve a relatively more difficult static position, this is quite intuitive.

As for Sayegh, the Viterbi search is complete and will find the optimal result allowed by the adjacent note assumption. How this scales up to extended pieces of music was not discussed. RD focus on evaluating their training mechanism rather than the pragmatism of their solution, and as such they prune the input music to remove repeated sections. It may be that computational hurdles were found in attempting to process the full pieces.

RD's experimental evaluation of their system is brief, based on the same data they had trained with, no evaluation is provided against unseen data. As such it is not clear whether their work is acquiring general rules or overfitting the data it has been trained on. This leaves room for further evaluation using a larger data set split into training and test sets, or using n-fold methods as described by T. Mitchell in [Mit97]. During evaluation RD suggest that "desired paths are not labelled consistently by the expert guitarists" is a possible reason for performance losses over split training sets ([RD04] Section 5). This variability is inherent to the domain and suggests room for further work on their model's robustness. It would also be desirable in further work to compare how their system performs for different musical styles, and to have solutions passed to human guitarists for qualitative evaluation.

RD conclude that their Path Difference learning approach has been successful. Given the presented evaluation data their paper presents a proof-of-concept for their learning technique but discounts it from system comparison. This is in line with TP's consideration of the work [TP06b].

Training to particular musical styles is an alluring concept. It is interesting to consider the notion in this domain, and whether algorithms could produce styled tablature. It should be noted that this notion of training to a style is not unique to a Viterbi based system, it could be incorporated into any system making use of a cost (evaluation) function by tuning that function to a particular style of arrangement.

A further work of the group is provided by Radicioni et al (RAL): [DPRL04]. RAL provide a comparatively detailed account of their encoding, though it is limited to monophony. They explain the concept of a matrix representation of the guitar strings, fretboard and fingers, and the notion of a DAG representing the problem search space. Unfortunately they do not go into details of moving from monophony to polyphony (where notes may happen at the same time) which could work with their encoding but would require non-deterministic sequencing of vertices occurring at the same time to explore all fingering possibilities, rather than arbitrarily deciding a particular note will be fingered first. This concept is seen again in Section 3.3 regarding [KCMT00].

A search strategy is not explicitly stated, but given their work's derivation from Sayegh's it is assumed they too use the Viterbi algorithm to find the least cost path. The focus of their work is on phrasal segmentation, as seen used by MY to simplify the problem in Section 3.1. Rather than as a simplification RAL are interested in the cognitive use of phrasing by a human musician. They believe, based on past research, that a musician will break the piece into segments and deal with them one at a time. Introspectively this seems agreeable, though not entirely correct. Their work's aim is to show that locally obtained optimum solutions tied together will perform better than a global optimum. Regardless of the cognitive aspects this phrasing does make the problem far more tractable, as it reduces the number of notes in the exponential complexity of the search space (described at the start of this section, and in Section 2.4). This assumption resembles the linearity property used in Linear time invariant (LTI) system analysis, such as signal filters [Opp97], and will now be referred to as *Phrasal Independence Assumption*.

It should be observed that although the pieces are segmented differently to Sayegh's work the limitations imposed are slight, given the Adjacent Note Assumption. The information which would be lost by assuming benefit which would be seen in a global solution has already been lost by not considering far enough behind in the piece for repetition. That is to say that for the systems presented by Sayegh and RD this phrasing work seems like a good addition, but is still prone to the issue of heterogeneity across chord movements. A proposal is made in Section 7.10 for an investigation of a segmentation approach which rather than keeping the entire solution in memory uses these phrases and remembers fingering shapes to try and reuse.

### 3.2.3 Genetic Algorithms

Use has been made of Genetic Algorithms (GAs) in this domain by TP [TP04, TP06b, Tuo06, TP06a, TPC06].

Following is a brief overview of GAs as an approach to optimisation problems; for a more formal and complete reference see Goldberg [Gol89], Whitley [Whi94] and Mitchell [Mit99]. A number of terms from evolutionary biology are used, these are defined in Appendix C.

#### What is a Genetic Algorithm?

A GA is a stochastic optimisation technique. It iteratively refines an initial population of possible solutions (referred to as individuals, genotypes within their encoding, or phenotypes when considered in context) to potentially divergent maxima through stochastic techniques. During each iteration it selects the best solutions available in the current generation, represented as a probability distribution where better solutions are more likely, and uses them to produce the next generation of individuals. This process continues until a stopping condition is met. The stopping condition is arbitrary, typically it may be the number of generations processed during

Figure 3.4: A Simple GA Run Flowchart

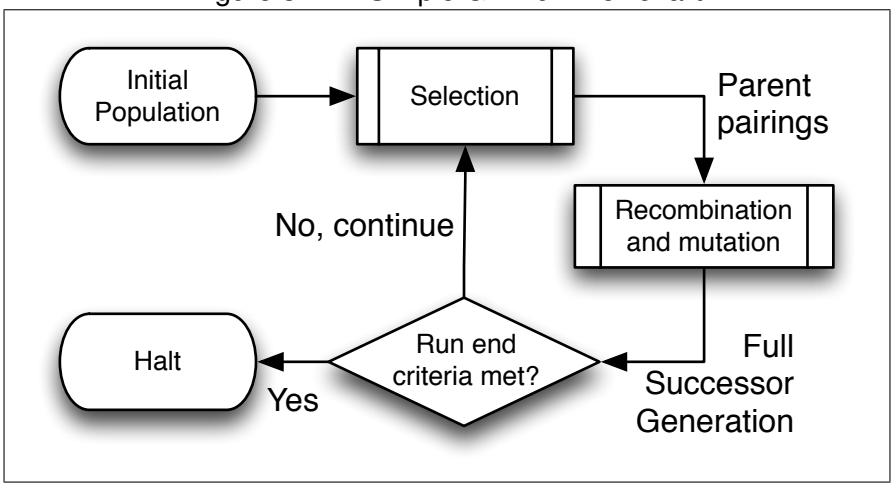
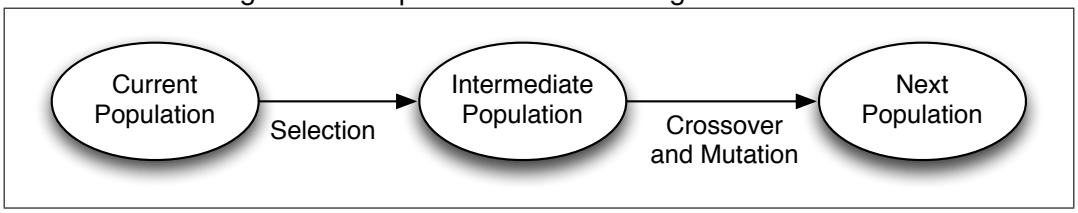



Figure 3.5: Population States During a Generation




a run. This process may be seen in Figure 3.4 and the changing population stages during a generation in Figure 3.5. An initial population will typically be produced by random seeding across the search space.

GAs are an analogy for biological concept of natural selection. The parameters, or features, of a possible solution represent the genetic data of an individual. This data is structured as a genotype structure consisting of chromosomes and genes. These terms are not used consistently in the field, though definitions may be found in Appendix C.

In artificial intelligence terms GAs are a general use ‘weak method’, as apposed to problem specific ‘strong methods’. Weak methods make few assumptions about the problem being solved [Whi94] and as such are general to many tasks [Mit99]. Although seeming an indirect and computationally demanding approach GAs have advantages and even the ability to solve problems beyond the reach of other optimisation techniques [Gol89]. A significant advantage is resistance to converging on local maxima rather than a global maxima for a given problem, this is due to the population being able to consider several competing good solution concepts at once. GAs use of a fitness function means they are measuring feature properties (or ‘payload’) of an individual solution, rather than some metadata pertaining to it such as a derivative. This is advantageous as some problems are difficult or impossible to determine derivatives for, making them unsuitable for such optimisation approaches. A computational advantage of GAs is that they scale well to parallel processing, and this may even improve their efficacy as discussed in the following section on applications to this domain.

An important disadvantage of GAs is that they are not admissible, due to the stochastic rather than complete nature of the search. For any given GA run there is no guarantee that the best solution found is the global maximum solution. The likelihood of finding a global maximum is proportional to the population size and the length of the run. That is to say the longer you process for and the more memory you use the better the result is likely to be. A consequence of

Figure 3.6: Example GA Data Structure: Rectangles by Height and Width

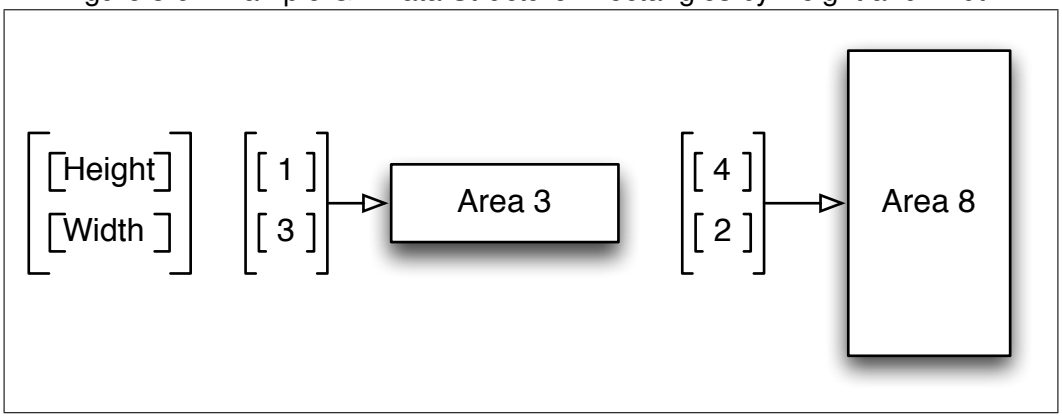
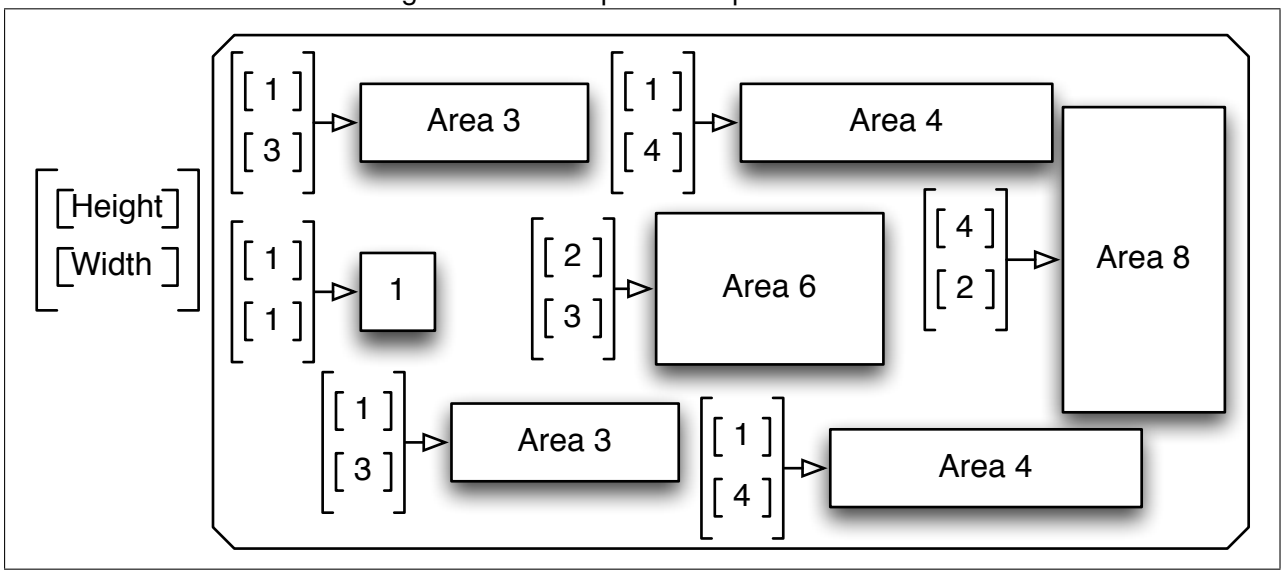



Figure 3.7: Example GA Population



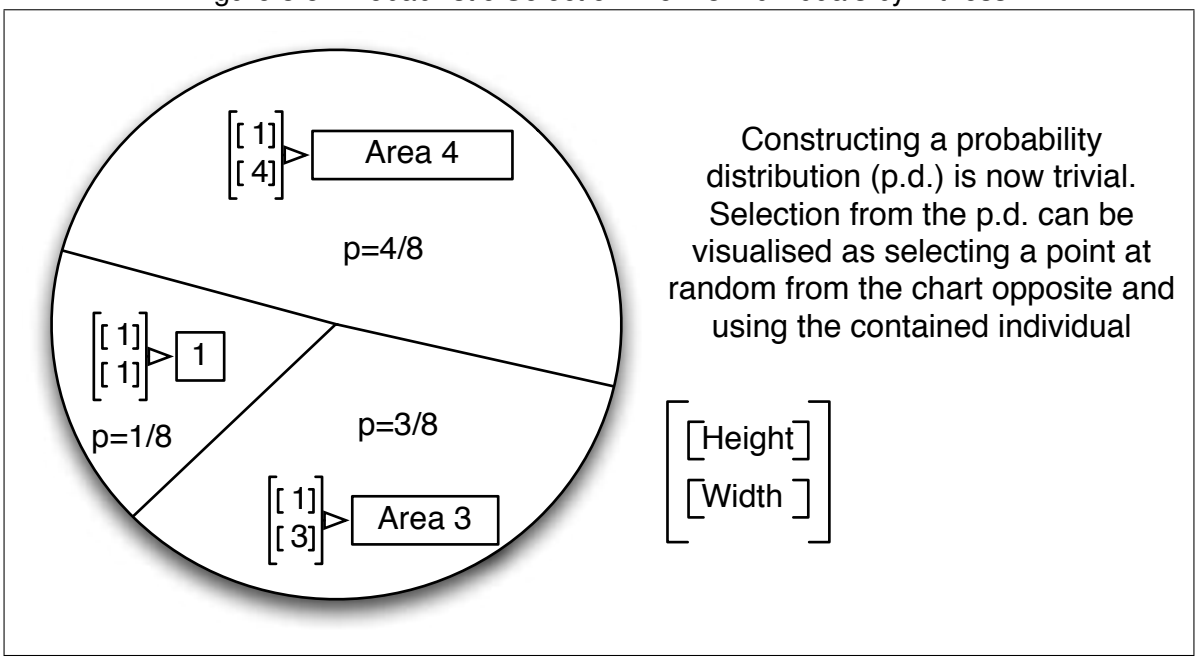

this is the stopping criteria. For a problem you may have an idea of a 'good-enough' solution, where you do not need a global maximum but rather a minimum quality. Once a solution meeting that criteria has been found the search can halt and return that result. Alternatively you may wish to run for a fixed time period and view the best results produced by that time. Once that time has elapsed you may decide to stop and use these results, to start again with different parameters and random seeding, or to continue the same run for a further time period.

Figure 3.6 shows a simple encoding of rectangles in terms of their height and width.

A simple GA may be used to optimise the area of the rectangles, where the height and width are limited to maximum values of 4. This corresponds to a 2-bit representation. For this example we seek out the rectangle with the greatest area, which is a square of length 4. Although a trivial example this will demonstrate the fundamental mechanisms involved in a GA described by Figure 3.4.

First of all an initial population must be created. This is randomly seeded across the search space. In this case a fixed number of rectangles will be created, each with a randomly selected height and width in the allowed range. Once this stage is completed the run loop is entered, where successive generations are created until the GA halts. An example population may be seen in Figure 3.7.

Figure 3.8: Probabilistic Selection From 3 Individuals by Fitness




Selection is the process of picking individuals from a population to use in creating the successor population. This process is stochastic rather than deterministic to increase the possible search paths that may be executed during a run. Fitter individuals are more likely to reproduce, but are not always the ones selected. It may be that taking a temporary hit to fitness will reveal a beneficial combination of features later on, for example there might be one wide rectangle with a low area relative to its peers of long thin rectangles. For selection to occur the entire population must have fitness values determined by a fitness function. A fitness value is intended to give an idea of the relative worth of an individual in terms of its surroundings, as such not having much meaning in isolation. A sensible approach to this is treating the likelihood value of each individual as its fitness function. In practice this use of concept and terminology are flexible as seen in Section 5.2.1. An example of 'Roulette Wheel' selection [Gol89] is shown in Figure 3.8.

Once parent pairings have been randomly allocated they produce offspring in a 'recombination' stage using a stochastic crossover operator or similar. Crossover typically results in offspring with a mixture of genetic data from each parent, but may also result in the parents being preserved unchanged. This recombination process can be arbitrarily complex, but is typically based on the manipulation of strings. An example crossover operation is shown in Figure 3.9 which selects a locus randomly and 'twists' all genes either side of it to produce the offspring.

When the 'next' or 'successor' population has been produced a statistical process selects a number of individuals to mutate, who then have a random part of their genetic data changed. For the fretting domain this would mean randomly re-assigning one of the values given for how to play a note.

Mutation is the random change of an individual gene to another allele, i.e. stochastically changing the value of part of an individual's genetic data to other valid values. Introducing this at a low rate presents a way to randomly reintroduce ideas about the solution, which are represented in the population's diversity, which have been thrown away. This prevents the stagnation of a population over a long execution time.

Figure 3.9: Random Locus Crossover Mechanism

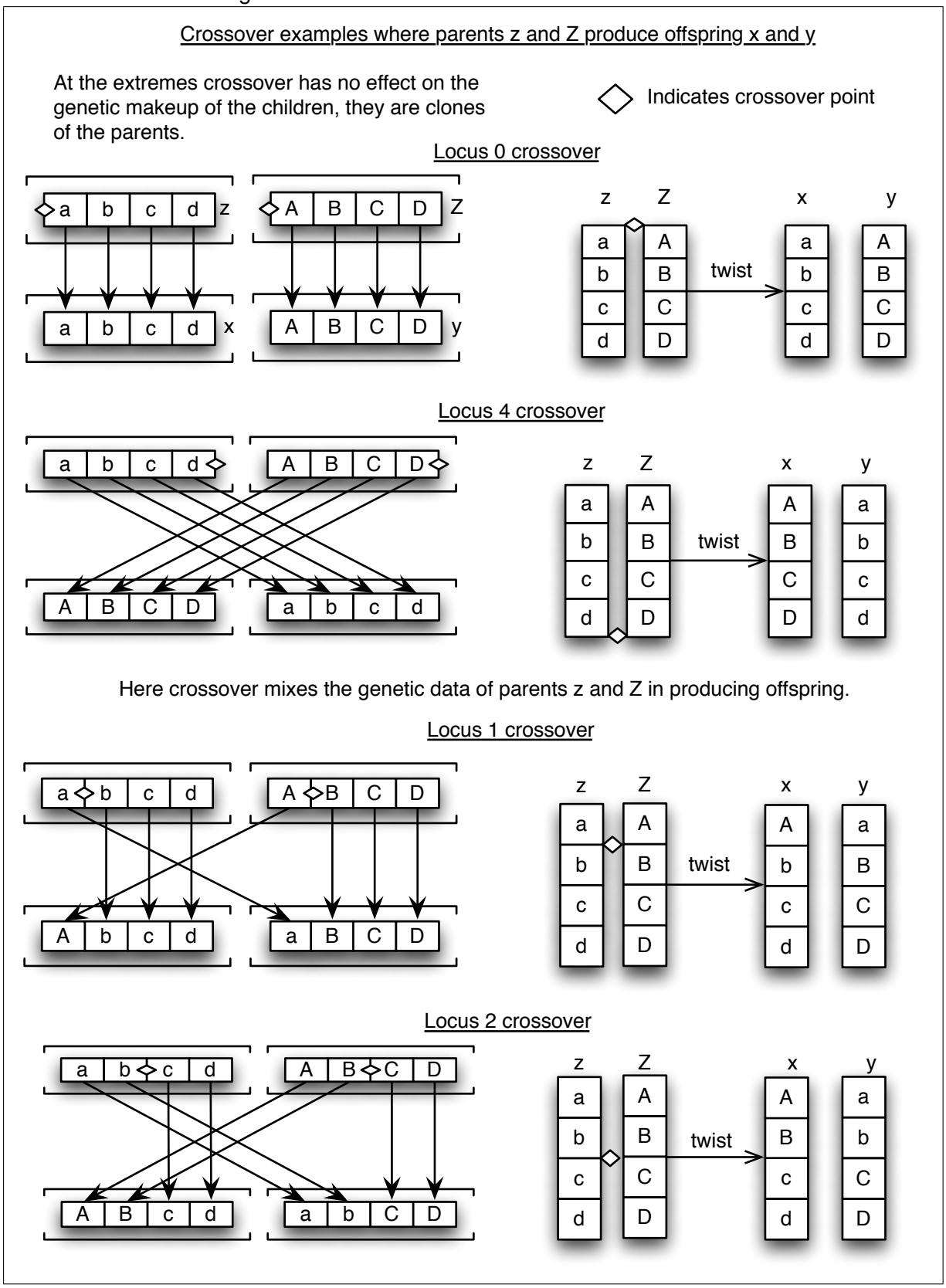
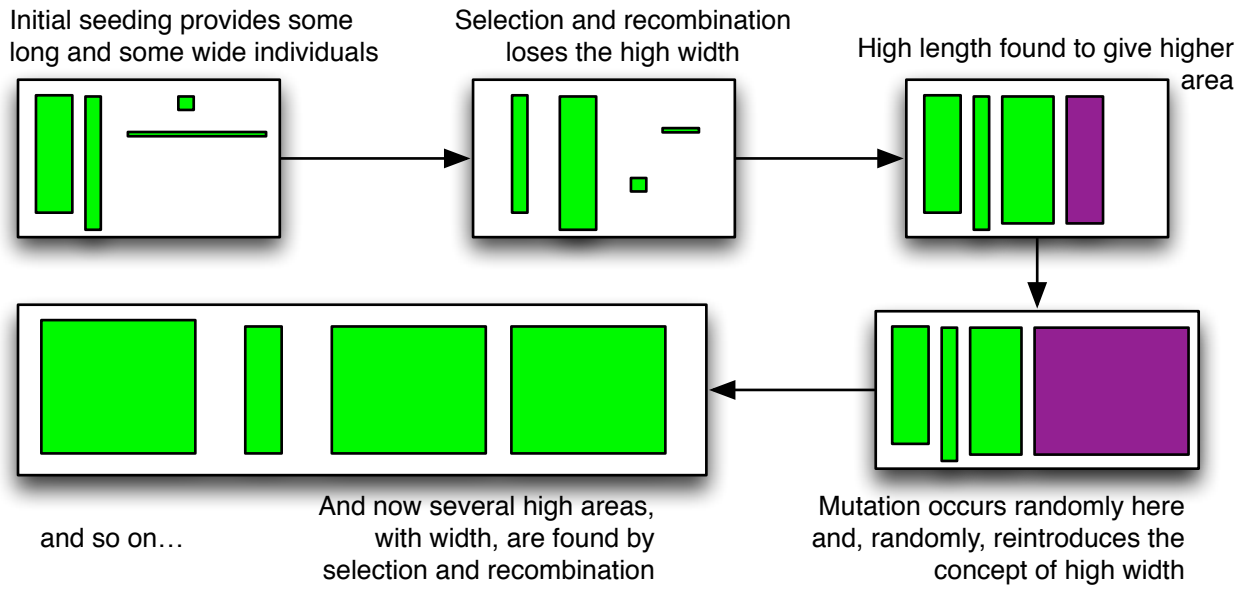


Figure 3.10: GA Mutation Example




Although mutation may seem unhelpful at first glance it ensures the robustness of GAs. Mutation may reinvigorate failing or stuck populations given sufficient time. Typically a GA is set to have a low rate of mutation, this is explored empirically in Section 6.2.2. Reproduction and crossover select and recombine the fittest genetic material, but due to the random selection process it is possible for mistakes to be made, and good genetic data to be lost. Mutation randomly varies the data in a population, which means helpful (and unhelpful) concepts may be introduced, regardless of whether they have previously occurred. Figure 3.10 shows a run seeking high rectangle areas which has become stagnant by losing the concept of high width. Through mutation the population recovers the idea and then by fortuitous selection and recombination produces fitter individuals, i.e. better solutions.

This summary has provided an example of a simple GA, though this is by no means a strict definition; the topology is quite flexible.

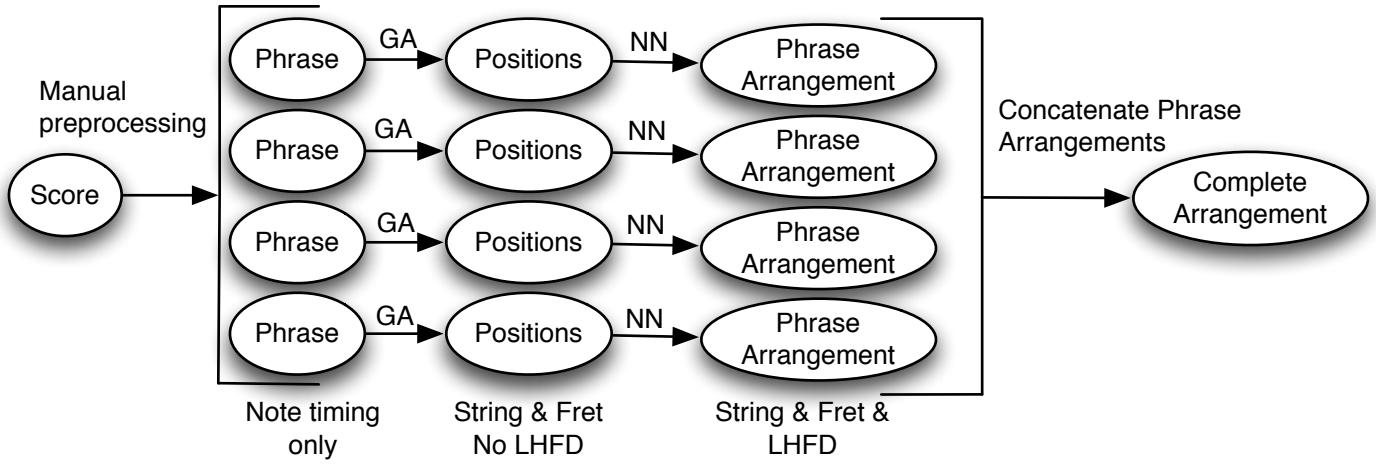
### Application of Genetic Algorithms to the Guitar Fingering Problem

TP approach the position determination problem (absent of LHFD) with a single GA [TP04]. An overview of their process is provided by Figure 3.11

TP manually pre-process their input scores into segments in the same way as RAL. A consequence of this is that they introduce the Phrasal Independence Assumption and are not working to find a global optimum solution. This is an important system design decision, and it is unfortunate that manual preprocessing is required.

TP's fitness function, analogous to those seen in Section 3.2.2, is feature-based. An important difference is that with a GA approach the Adjacent Note Assumption is not required, so their evaluation function has more information available. As mentioned, however, the Phrasal Independence Assumption is a limiting factor which is not thoroughly addressed in their work. Introduced in two sentences it is justified by computational tractability for long pieces of music, with no mention of cognitive benefits such as in RAL's work. No discussion is provided of the negative consequences of the assumption on the search space or results obtained. Not

Figure 3.11: Overview of Tuohy & Potter's Work on the Fingering Problem




treating a piece as a whole may lead to important inconsistencies in re-used shapes such as chords: a guitarist will be displeased if they are told to play the same notes in two or more different ways during the same piece of music. Further consideration of this issue can be found in the discussion of RAL's work in Section 3.2.2.

A training stage is used similar to that of RD, though a GA is used as a 'meta-GA' to train the cost function by executing the main GA many times to optimise the cost function parameters ([TP06b] Section 4). The cost function itself considers the work of HM and others, but is stated to be mostly introspectively heuristic.

Although a feature approach is similar to the works seen in Section 3.2.2 the absence of LHFD is a considerable abstraction.

LHFD is later added [TP06b] in an additional processing stage, fingers are assigned to their tablature by a Neural Network (NN). This is an interesting concept and seems sensible, given that the fingering positions may be interpreted as shapes to recognise and associate fingers to. It will however be largely dependent on its training data and would need to be carefully evaluated. A NN will produce results at a greater speed than a GA for this stage. An issue that must be made clear with this two stage approach is the absence of LHFD in the position determination stage is a considerable abstraction. TP consider it difficult to track the motion of fingers

No mention is made in their work of embellishment techniques such as glissando slides. Despite this it is reasonable to believe that they can be incorporated into a GA through encoding or fitness function metadata, which can be arbitrarily complex. Increasing the encoding complexity will notionally increase the size of the search space. TP make use of a distributed GA technique [TP06b] analogous to isolated island populations where the best individuals migrate to the other islands periodically. This structure is ideal for parallel computation, opening up a range of options for threading, batch jobs and similar for execution on several platforms concurrently.

Unlike RD, TP do not attempt to induce style from their training data, which poses issues of generality and the stylistic consistency of their training data. These issues are not discussed in their work. It may be that this is less of an issue for positions than for LHFD, which TP do not determine in this stage, but this seems unlikely.

Their training set is larger than that used by RD, 30 pieces opposed to 7, but may still be small given the size of the search space. Having not presented the cost of training and the perceived benefit as the training set increases in size this is hard to assess. This approach

is preferable to manual tuning but care needs to be taken in selecting the training data and in testing and interpreting the result.

TP's evaluation aims to demonstrate an ability to cope with music which caused errors in other available systems. They also consider the work of RAL and claim to have bettered it, achieving marginally superior results on RAL's test data for positions, and claiming that their own independent test data was more difficult to produce tablature for than RAL's test set. A conclusion is also drawn that their distributed GA approach produced far better results than a simple GA, due to the nature of the domain and the number of competing solutions which a population may develop into.

## 3.3 Also of interest

An aside of interest is work by Traube [TS00] on determining the plucking point, and hence the fret, of notes in audio. This could provide the basis of a closed-loop automatic transcription system, using a playable-tablature algorithm to help with noisy segments, or conversely the audio data could be used to provide hints to a tabbing algorithm.

Kim et al (KCT) worked in a different area [KCMT00], 3D modelling and animation of a violinist's hand, but covered a similar problem. They were interested in the detailed mechanics of left-hand motion, as appropriate for their problem. Still, they take as an input the same musical data, without fingering positions, as the guitar fingering algorithms. This is processed in a similar manner to produce wrist and finger positions to animate. They use a best-first search to find fingering positions, determine the position of the fingertips (which will be applying pressure to the strings, to actualise a fingering/fretting position), then use a NN to find an optimal wrist position for playing. If this is feasible they use it, otherwise they start again with another fingering position. This fingering position approach derives from HP's work on animation and control theory [HvdP96]. It assumes that a sub-optimal solution is acceptable so long as it is possible to play. This is reasonable for feeding into an animated simulation, but not good enough for a system intended to provide solutions for human musicians. Their work on sound quality does not seem relevant for a guitar, as guitar frets obviate the issue of undesirable sliding by fixing the string lengths at discrete positions, on the violin the playing positions are continuous and slight movement will change the string length, altering the pitch. The ease of execution measure is that of Sayegh [Say89] and further justified by stating that by playing in a similar region of the neck, the tonal qualities of the notes' harmonics will be more consistent than moving around a lot.

KCT state as advantages over a rule based approach (e.g. Section 3.1) "adaptability to different factors, ability to consider global effect, and scalability to polyphony". While the first two statements are vague, the latter is linked with an intriguing approach they take to moving from monophony to polyphony. The notes required in the chord at a given time are "serialised" and assessed with their monophonic algorithm, as if they were to be played in sequence. This is certainly a simple way to bridge the gap, however as an implementation issue it would be necessary to consider all notes of the previous chord if using a read-ahead (or behind) technique, meaning a set is considered rather than an individual note as in monophony.

[HM02] investigated the complexity of left-hand use in classical guitar playing from a biomechanical and behavioural perspective. Although relevant the paper demonstrates the complexity of the issue at hand rather than providing results or conclusions from which a definitive cost function can be derived. This is unfortunate since many of the papers presented in Section 3.2 treat it as justification for their cost functions.

It may be of interest to consider approaches seen in other research domains to the problem.

A technique seen in ASR systems is storing feature metadata, such as derivatives of an audio signal. Storing additional feature information is able to offset the Adjacent Note Assumption to an extent. The limitation of this metadata is that you cannot make use of information you have not stored. For a general representation where many different evaluation functions can be applied to the arrangements the full solution will need to be available either in its original or a compressed form. Another interpretation of LHFD may be that it compresses the full arrangement into a smaller representative approximation, in a similar way to cepstral coefficients used in speech processing to represent a speech recording more concisely. This may provide a 2-stage solution where appropriate LHPs sequences are determined first, then these are expanded into full solutions and perhaps developed further.

## 3.4 Conclusion

Common themes can be identified in these approaches: high level approach, phrasal segmentation and other simplifications, musical complexity, performance features, musical style and algorithm training.

As seen the two main approaches are the expert system and the optimisation approaches. The optimisation approach, finding the least cost path or 'optimum path paradigm', has yielded a number of more sophisticated systems and a higher quality of resulting publications. Works from the expert system approach, although more limited in scope, do still offer insights and suggestions for a basis of work that are useful going forward. For this work the optimisation approach has been selected.

Segmentation assumptions are an important theme across the works presented. They make the problem more tractable at the cost of available information for problem solving. A consequence of this is that the the function approximation provided by the algorithm is noisy, in the sense that its results will sometimes err. The Adjacent Note Assumption used by Sayegh and derivative works considerably reduces the search space which needs to be considered at any time, though as stated by TP important information is lost which a guitarist would use to determine an optimal arrangement. It may be possible to offset this problem by maintaining a record of the LHP through the piece. For more complex evaluation functions making use of cognitive aspects this will not be sufficient. An example of this issue can be seen in chord use: a mediocre arrangement according to heuristics such as finger movement and hand span may become the most desirable arrangement if it sees the re-use of familiar chord shapes throughout the piece, this would be hidden by the Adjacent Note Assumption where the chords were not adjacent or tracked in some other fashion.

Another facet of segmentation explored by RAL was using the Phrasal Independence Assumption to optimise the piece in individual segments rather than finding an optimal solution for the entire piece. Although more in tune with how a human would approach the problem it does discard some information available to a human during this process. A guitarist may use knowledge of other segments to make a more comfortable or cognitively simpler arrangement, which may be less efficient viewed as an isolated segment. The question may also be asked: is a human solution the best we can do? It is undeniably a good starting point, but why limit ourselves to that when more information is available that a human does not use. Evolution has not developed human beings to perform well at this task as it has speech, hearing, and other human faculties.

This idea of global and local optimisation has not been explored in depth with a GA approach, nor has the storage of metadata been considered as a way to bridge the linearity assumption's loss of knowledge about other segments. Although TP's work describes issues

with the Adjacent Note Assumption they go on to introduce the similar Phrasal Independence Assumption without acknowledgement or exploration. Segmentation and independence assumptions are important design decisions and should be clearly stated in any further work. These decisions are stated in the requirements chapter (4) and where appropriate during experimentation.

Polyphony and LHFD have been treated as either intrinsic or additional problems in different papers. A possibility has been suggested of breaking polyphonic sequences down into a pseudo-monophonic problem (3.3), but at the cost of introducing non-determinism and further increasing the search space (3.2.2).

Embellishments such as slides and string bending have largely been ignored in the works presented. MY introduced some of these concepts in their work but in an undesirable fashion.

Tailoring a system to differing musical styles (as initially suggested by Sayegh [Say89]) is an interesting notion, but is unlikely to be successful without a well developed cost function and the use of a machine learning approach over a large selection of data. These issues were encountered by RD in their attempt at such a system [RD04]. It is not considered further in this work. This system is intended to be genre agnostic, beginning with musical scales and advancing into beginners pieces as the system evolves.

# Chapter 4

# Requirements and analysis

Chapter 3 presented two groups of published approaches: Expert Systems (3.1) and the Least-cost Path Optimisation approach (3.2). Of the two the the optimisation approach appears most promising. Important divergences within this track were identified as the Adjacent Note Assumption and Phrasal Independence Assumption.

The system proposed herein is a genetic algorithm which makes neither of these assumptions and includes LHFD. This sets it out from the crowd as it addresses the problem in one pass without simplifying assumptions, thus making the entire solution available to a cost function. It is then up to the cost function to make use of this information or not. Pathological and real-world examples will be used to investigate the importance of the information which simplifications would lose.

Working with these constraints has important consequences. Positively, the algorithm can work in a fully automated and self-contained fashion. No manual preprocessing of input or output data is required. Negatively, the resulting search space is vastly larger than of previous works, in fact it will be exactly that considered in Section 2.4. This may prove to be a naive decision, but is divergent from the works presented and if nothing else will reinforce their decision to *manually* split the pieces up before processing, rather than having a fully automatic solution. Manual pre-processing is clearly undesirable and investigating its necessity worthwhile.

The computation tractability of this system should help to understand and justify the segmentation approach and its shortcomings. Work will begin with this full representation and an empirical evaluation of working in this search space made. A successful result will be either showing that a GA is able to converge on good results reliably in this larger search space, or to give an indication of some limit to the length of piece for which this is possible. The latter result may provide a useful guide for upper segment sizes in future work. Cost function aspects are considered and implemented, but a full study of their implementation is expected to require further work.

The aim of the project as a whole is to produce the GA, a suitably effective cost function, and to evaluate the system's performance with real users. Given the complexity of these tasks the project will be produced in an incremental fashion, reaching results for the simpler monophonic problem first and then being refined to polyphony and other features.

## 4.1 Domain Encoding & Evaluation Function

A representation should be developed of the musical score and associated fingering solution. This will need to track note name, start, duration and octave; string and no fret position; left-hand finger used.

LHFD is to be included as a fundamental part of the encoding. Monophony is sufficient as a starting point, but the design should be extensible to polyphony once the framework is in place and results obtained. The system should be implemented in such a way that this transition from monophony to polyphony involves changes to the data representation and cost functions only.

The system must correctly discriminate between notes with the same name but different pitch within an arrangement, notes in different *octaves*. The interval between these notes must be maintained when producing tablature to play them with.

As the implementation evolves cost functions should be developed to take advantage of the available information. A template should be developed allowing for further cost functions to be determined in a simple manner allowing them to focus on their specifics rather than the GA as a whole. It should be made possible to run the algorithm with different cost functions in a simple manner.

Empirical cases should be set out for several facets of the fingering problem and issues with the Phrasal Independence Assumption and Adjacent Note Assumption. Solutions to real music should also be devised and their quality assessed by third parties. Comparison with a body of tuition material should also be made.

An optional feature is to attempt a cost function which rewards repetition or reuse of patterns in a solution, for example playing a chord in the same position with the same or similar fingers.

## 4.2 Evaluation

Issues arise when evaluating system performance against human experts, as explained by TP in [TP04]. Care must be taken with the assumption that a human tablature producer will behave in a rational and consistent fashion. Indeed given the human condition and diverse nature of solutions in this domain it seems unlikely that they will producing the most-optimal general solution each time they attempt the task. While this may be advantageous during training it does not lend itself to one-to-one evaluations of results between system and expert.

Given this range of possible answers even if a large enough test set were found to evaluate the algorithm with it may be that the system has in many cases produced divergent but equally valid results. As TP quite observantly state it is difficult to assess which tablatures are better than others, as this is the same problem addressed in fitness function design. It may be decided that performance against a standard test set is a good way to compare divergent algorithms, however the varied nature of arrangements must be borne in mind, perhaps allowing several good solutions for each problem score, and having some method to determine how close a fit the algorithm's results are to these model answers.

A simpler robustness metric, and thus method to evaluate the algorithm, is to qualitatively examine whether produced tablatures are considered fit for purpose by guitarists. This approach was taken by TP [Tuo06] and a sight-reading exercise was performed by Emura et al [EMHY06]. Emura's approach experimentally observed performance events such as mistakes made, which lent themselves to a more quantitative assessment. This also could form part of a general test set, though a large sample of performers would be needed to reduce the random elements of individual variation and sample selection errors.

A qualitative assessment can also be made of the produced tablatures. Where and why did they fail, and were these failures by the algorithm or the producer of the test data? Was the human expert consistent, did they make mistakes? Online tablatures are rife with transcription errors and these account for some of the data used. The notion of a best or optimal tablature

may be misleading, there may be divergent alternatives which are ranked apart only because of the granularity and approximations inherent in the cost function rather than a true reflection of their playability. Receiving this feedback could trigger a refinement loop leading to a production system.

The following outlines a brief user study which could accompany the work, were the system to reach an adequately sophisticated level. As previously discussed empirical validation should be sought out not only of machine learning characteristics of the algorithm, which are not in themselves a solution to the problem, but also of the value of the solution. Do real guitarists, of a range of abilities, find the system productive? Do they disagree with it? How do they differ from it, and why? What would they do instead?

A survey approach of a small sample, 5-8 guitarists of varying ability, should provide some insight into the success of a produced LHFD augmented tablature generator. This test should involve giving the guitarists tablature samples and sheet music which have been processed by the algorithm, and asking them to play the pieces, suggesting any modifications they would make to fingering (but not to the music itself).

The guitarists should also be given an opportunity to use the algorithm themselves, or in cases where that is not practical to provide music for it to arrange into LHFD augmented tablature form. This will ensure that the test data is arbitrary, as the users could pick anything suitable. Suitability should be determined by whether the music can be accurately encoded for the algorithm to process, not by whether the experimenter feels their algorithm will fail to produce good results for the score.

## 4.3 Implementation Functional Requirements

The system must scale to execute on systems with differing amounts of runtime memory available, in the range 512MB-2GB or above. These are reasonable limits at the time of writing.

Cross-platform operation of implementation. Must run on Unix systems with ssh available for remote job distribution. Intended targets are Mac OSX 10.5 for development and testing, and Ubuntu and CentOS Linux for experimentation (the latter on the University network & ExpressTrain grid engine). Given the standardised nature of Unix applications (in particular ssh, standard IO, and shell scripts) this should be trivial so long as a commodity programming language is chosen for implementation.

Microsoft Windows systems are not deemed suitable targets due to the incompatibility of terminal services and general absence of freely available tools and services. That said given the ubiquity of the platform it would be beneficial for reader demonstration purposes if the implementation also worked in a Windows environment. As such a Windows implementation is an optional feature.

## 4.4 Implementation Non-Functional Requirements

A user should not have to build the software from source, and should have to install a minimum of third party applications. Source code should be made freely available for review and modification.

As this is not a production system execution time is not an issue, so long as experimental results can be obtained within a reasonable timeframe. Any limit set here would be arbitrary and as investigating this is an aspect of the research no limit is defined. In a production system the processing overheads of a GA approach may be moot. A program can be left to mull over the data for significant time as long as the user gets good results. The processing time given to

a GA is flexible. It could be calibrated by the end user, as well as having special considerations for a user interface showing current results and options for halting, continuing, etc.

Producing code for a modular and reusable GA is considered beyond the scope of this work. While coding may naturally progress in this fashion and refactoring to a general solution may later be possible, it is felt that there is no inherent need to approach the implementation as such at this stage. Indeed the focus of this work is on the problem domain and evaluation functions rather than some novel GA design or feature. In this work a GA was implemented from the bottom up as a pedagogical exercise for someone new to GAs, but it would be a valid approach to make use of an existing GA framework.

## 4.5 Omissions and Further Work

### 4.5.1 Segment Arrangement

A useful continuation from the optimisation work is to investigate an algorithm which takes locally optimal segment solutions and finds the best global solution when fitting them together. Again this could be done by numerous approaches, including an expert system which modifies the solutions while piecing them together. A distributed GA approach may also be of interest.

The task of splitting a musical score intelligently into shorter segments or phrases, the segmentation problem, is beyond the scope of this paper, but a study of work in the area would be useful in extending this system.

### 4.5.2 Style

Given the treatments which have been provided previously it seems that style is a desirable training feature in a solution to the guitar fingering problem. It is, however, far from settled; no thorough evaluation of the success of previous algorithms has been provided.

The issue of style within tablature generation is a complex subject with no clear solution. In producing a system which can be modified to use different cost-features as desired by the user this issue of style is eschewed: it enables a third party to work on this at a later date.

The domain can be addressed generally rather than thoroughly reviewing formal techniques for one of many styles; indeed with the treatment seen in these papers the material does not appear to be used at all, or is used ad-hoc in heuristic evaluation methods. The fingering problem is still interesting in the absence of a formal definition of playing styles. A point in support of this is that the majority of styles do not have formal rules for positions or LHFD, these are seen in trained styles such as Classical, Flamenco and Jazz, but not in the more bohemian styles such as Rock and Metal. Self-tuition as opposed to formal training, and the age of the respective styles are possible reasons for this.

### 4.5.3 Embellishments

Embellishments such as glissando slides have been omitted from the initial work, as their presence will add considerably to the problem complexity. A misleadingly simple implementation where embellishments are coupled to note information will work for monophony but not scale up to polyphony with overlapping note starts and durations. It may be helpful to consider a data representation where graph vertices represent notes and graph edges may have different forms for standard note changes, slides, hammer-ons and so on.

### 4.5.4 Evaluation Function Study

The cognitive and biomechanical evaluation of guitar playing produced and used by the papers presented in Chapter 3 leaves much to be desired, and has perhaps been exploited in its vague nature to say whatever was convenient for researchers in their cost function productions. Further work investigating this area and attempting to produce a standard cost function, or further developing and contrasting alternative models is desirable. A divergence in this area would centre around the Adjacent Note Assumption, which given its relation to the search problem would likely not be considered by studies from disciplines other than computer science.

### 4.5.5 Range

An important concept in restricting the set of possible solutions is the natural frequency of notes on the sheet music stave. Although a trained Classical guitarist is constrained to an expected range (low E, below the third ledger line under the treble clef is the open bottom E string) this is not true of more bohemian guitarists, who may be interested in seeing and perhaps trying each alternative to find something unusual.

It is important to differentiate the range of the piece from the concept of octaves within that range. The concept presented here is to transpose the entire piece of music up or down the octaves, not to allow the relative intervals between the notes to change. That is to say that for the fundamental frequencies any two notes of the piece $x$ and $y$ transpose into $x'y'$ such that $x > y \rightarrow x' > y'$ and $x < y \rightarrow x' < y'$. For example taking E and A chords played at the open position (0th fret) and increasing each fret by 12 will increase the pitch of each note by an octave.

Where the intervals between notes are maintained across the entire piece, but all notes are changed, the piece is said to be transposed. This is not uncommon, pieces are often transposed to suit a singer's vocal range. This could be incorporated into the system by means of a pre-processing stage which finds alternative transpositions of the input score and processes them separately before coalescing the results and returning the best ones found. This would be a trivial to implement given the rest of the system and has been omitted.

### 4.5.6 User Configuration

All papers either consider standard EADGBE string tuning or do not specify it, with no ability mentioned to vary this dynamically. Systems with modifiable evaluation functions, be they obscured Hebbian learning structures such as a NN or more clearly tuneable functions, could be trained by the end user to meet their particular requirements.

Doing this in a user-friendly manner would be challenging, but could borrow ideas from applications such as voice or handwriting recognition where a user provides personalised training data to improve performance for their particular use. In this case the algorithm would be trained to use chords and finger patterns or hand movements preferable to the user.

This work requires a system already in place with a sufficiently sophisticated evaluation function to train to a particular user. As these are not currently in place this is left for future work.

### 4.5.7 Audio input

By means of a frequency tracker and mapping frequencies to notes it would be possible to to produce tablature from some audio recordings, largely dependent on the noise levels and

possible filtering. This would provide another means of input, and have little to no effect on the rest of the algorithm. A system attempting to discover positions from audio data is mentioned in Section 3.3.

### 4.5.8 User Interface

There is a lot of scope for Human-Computer Interaction work with this system's input, process control, and output. Aside from the impracticality of designing a sophisticated interface for an incomplete experimental system this seems a sufficiently complex issue to be the subject of its own project. A simple interface may be produced for demonstration purposes but is not considered relevant to the study.

# Chapter 5

# Genetic Algorithm Design and Implementation

This chapter addresses the design of the Genetic Algorithm (GA) used herein to address the guitar fingering problem.

As presented in the literature review (3.2.3) there are a number of issues pertaining to GA design which must be addressed: the structure of the solutions as genotypes within the GA; the reproduction method, including selection, crossover and mutation; and the statistical analysis of different runs of the algorithm.

## 5.1 Structure

A high level interpretation of the code produced is presented here, though some terminology could be restructured to obtain the same result. In brief each gene represents a feature, alleles are sets of possible feature values for particular genes, chromosomes are structures containing genes. Genotypes represent an entire individual from a high level, while chromosomes are related more to the structuring of genes. In a simple structure the terms genotype and chromosome may be used interchangeably. Terms are defined in Appendix C.

Each solution is a genotype consisting of a chromosome for each note in the musical score, as in Figure 5.1. This is appropriate since both the chromosome structure and musical score are identical for each genotype. Each different score considered will produce a different genotype structure. Each chromosome represents the fingering position for an individual note in the

Figure 5.1: Example Genotype Structure

| Feature | Note Chromosomes | | | | |
|---|---|---|---|---|---|
| Common Invariants | | | | | |
| Start | 1 | 3 | 5 | 7 | 10 |
| Duration | 1 | 2 | 0.5 | 1 | 4 |
| Name | F | G | D | C | E♭ |
| Octave | 1 | 1 | 2 | 2 | 2 |
| Genes | | | | | |
| String | Low E | Low E | A | D | G |
| Fret | 1 | 3 | 5 | 10 | 8 |
| Finger | 1 | 3 | 3 | 1 | 2 |

piece, having genes for fret position, guitar string, and left hand finger. The chromosome itself carries invariant knowledge of its named note, octave, start time and note duration. These are alike for all genotypes, representing the common musical score. This data being identical for all individuals presents a possible memory optimisation, storing the data once and looking it up rather than storing it as part of each individual data structure. In this work it is implemented in Java using shared object references, so the data is stored once and referenced from many places.

The guitar fret gene is an integer representing the fret played, or 0 for an open string. The guitar string gene has 6 discrete alleles, representing the strings of the guitar. These are strongly coupled with the guitar fret as the two determine the musical note being played. The implication of this is that for a given musical note there is a finite set of allowable string and fret gene combinations. This is important when determining random values during initial population generation and mutation. Here they have been presented as distinct genes, but given this coupling it is also sensible to treat them as a single two-dimensional gene, analogous to complex numbers storing magnitude and phase information. As they will be handled at the object level holistically this will make no difference to the GA reproduction mechanism; it doesn't see the contents of the genes, only manipulates the genes as a whole. Of course the cost function will need to understand how these values are stored, but ubiquitous object oriented programming paradigms, such as interfaces, obviate this issue by hiding implementation details from the rest of the algorithm.

The left hand finger gene is free to take any allowed value in the assumed range 1 to 4, where 1 is the index finger and 4 is the little finger. During implementation an additional value was found to be necessary representing no finger being used when playing an open string. This can be added simply by extending the range to 0 to 4 where 0 is no finger used, but must unique to fret 0, and also the only finger allowed at that fret.

As previously mentioned other representations are possible, and may prove to be important in scaling up to polyphony. At present this seems sufficient, since all pertinent data has been stored in an accessible form.

## 5.2 Reproduction Method

In a Simple Genetic Algorithm [Gol89, Whi94] one generation in a GA's execution consists of the current, parent, population being used to create an intermediate population through selection. This intermediate population is transformed into the next, child, or successor, population by recombination and mutation.

In the presented algorithm there is no intermediate population, rather parents are randomly selected from the current population and the next population is produced directly, 2 individuals at a time. This removes the need for a second iteration across the population. Crossover is used as the recombination method.

It was decided to maintain a fixed population size, as this was simple to implement and control, and allows for control of algorithm efficiency and computation cost trade-offs.

### 5.2.1 Selection

To produce a successor population first an existing population is ranked by fitness and a probability distribution (PD) produced reflecting these fitness scores. Parent pairings are selected from the PD to create the successor population with. Where an odd population size is required one of the successor population members is discarded at random after the oversized

population has been produced (as suggested by M. Mitchell [Mit99]).

The fittest individuals in the population are to be assigned a higher selection likelihood. In a fixed size population the number of times an individual is expected to produce offspring is

$$\textit{Expected times selected} = \frac{\textit{Individual's fitness}}{\textit{Population's average fitness}} \tag{5.1}$$

This is an analogue of what biologists call "viability selection" [Mit99].

A PD for the population is produced using Goldberg's roulette wheel metaphor [Gol89]. An implementation of this is discussed in Sec 5.3 and shown by Equations 5.1 and 5.2.

Selection "with replacement" [Mit99] has been chosen, meaning that each genotype may be selected multiple times as a parent when creating a successor population. This not only simplifies the roulette wheel by allowing it to remain static for a selection process, but it allows individuals to fulfil their viability without restriction.

Due to the stochastic nature of this selection process low likelihood individual will be selected occasionally. Exploring these apparently undesirable results enables the GA to search a larger portion of the search space. Only pursuing fit individuals could cause convergence on local maxima within each run, though each may converge on different maxima; this is explored in Chapter 6 The chosen selection method will subsume these local maxima and combines them with other concepts about solutions (which are encoded in the genotypes, [Gol89]). This results in good ideas from a local maxima being mixed with ideas about other regions of the search space, and so broadens the search to find other, potentially superior, maxima. Because of the stochastic nature of the process, however, a large number of selections is required for this to occur, which is dependent on the population size and the run length (number of generations produced).

An alternative to this method is provided by mutation, though it was decided not a suitable replacement, rather something to use as an additional tool to (given enough opportunities) reinvigorate a stagnant population with new concepts.

### 5.2.2 Crossover

Once parents have been selected the pairs are recombined, each pair producing a new pair of offspring to form part of the successor population. Recombination consists of applying a crossover strategy to the pair, which will determine whether and how to apply crossover to the pair depending on the strategy in use.

Two examples are suggested by Goldberg and M. Mitchell. Both involve a likelihood parameter $P_{crossover}$ but make quite different use of it. A fixed recombination point crossover performs a single random check to find whether crossover should be applied, and if so recombining the pair about the central locus point. An alternative is to stochastically selects a locus to split each pair about. This is done by traversing the chromosome from first to last locus, stopping when $P_{crossover}$ is first satisfied and using that locus for crossover. This may result in crossover effectively not occurring when the first or last locus are selected, since there are no genes preceding or following them respectively. An example of the latter is provided in Figure 3.9.

The effect of these strategies on the algorithm's results are compared in Section 6.2.3.

### 5.2.3 Mutation

Both Goldberg's and Mitchell's suggested models apply mutation to each chromosome with likelihood $P_{mutation}$. This may be performed within the crossover mechanism or applied afterwards to the entire population. The former will save an iteration over the entire population,

while the latter may result in a more clearly partitioned algorithm. The efficiency gain seems more important so long as the code is working, given the large population sizes to be worked with (expected to be $10^3$ and greater).

The mutation method selected should be able to help reinvigorate a stagnated population in the aim of finding superior solutions. This method will modify individuals a small amount rather than replacing their entire structure with a new random individual (an alternative mechanism). This is thought to be good as it will still inherit good fitness from its parents, with small changes from mutation, and thus stands a good chance of being carried forward into successor generations. This is also more in agreement with the notion of mutation, which is for small changes to occur, rather than massive changes. The performance of this choice is evaluated in 6.2.2.

### 5.2.4 Algorithm Summary and Implementation

In summary we have described a genetic data structure suitable for encoding monophonic musical scores and associated fingering solutions. This structure will be populated with random initial values within the search space to form an initial population. This initial population will then enter a run loop, where until a stop criterion is met new populations will be formed by probabilistically selecting the best solutions and breeding them in the hope of producing better solutions. Mutation is present to prevent stagnation. The following section (5.3) describes various cost functions used to determine solution fitness.

Implementation of such a system is a routine exercise in data representation, manipulation, and random number generation which may be completed in numerous languages and paradigms. The selected language was Java, given its cross-platform features meeting the functional requirements presented in Section 4.3. Its compiled (to an abstract machine form or 'bytecode') nature will also offer performance advantages over interpreted alternatives such as Ruby, Python and Perl; indeed it is claimed Java offers performance similar to, and in some case in excess of, natively compiled C++ applications [CH05]. Later on in implementation it was realised that a purely functional language such as Haskell would be an interesting approach to the problem given the stateless nature of the algorithm.

The development method used was a mixture of bottom-up experimentation with unfamiliar GA mechanisms and a top-down Agile approach inspired by Behaviour Driven Development used in industrial[1] Ruby-on-Rails web application development[2]. This practice is an evolved version of Test Driven Development (TDD) where more emphasis is put on tools which present tests as executable specifications which may be comprehensible to a non-technical user or client. This is unimportant for this application and as such was not adhered to strictly, nor were suitable tools sourced for doing this with Java. Instead standard TDD tools JUnit and JMock were used to produce code of a demonstrable quality and robustness to change [Bec04]. The practice adopted was to write tests before code, as typical, and to write the tests in such a way that the test function names represent specification clauses, clearly explain the intent the test is verifying, rather than simply the name of the method they are testing. Again this was loosely adhered to so far as was useful during development. Version control software (Git, as used for Linux kernel development) was used to track changes during the project, and both the source code and the revisions history may be explored online via web interface at `http://github.com/nruth/fingar`. Instructions for obtaining and executing the code are included in Appendix F. A pitfall discovered during testing was false positives generated due to the stochastic nature of the algorithm under test in several places. This may be alleviated by strategically repeating segments of tests and ensuring that they pass or fail on one or

[1] `http://www.ardes.com/` amongst others

[2] See `http://behaviour-driven.org`, `http://rspec.info` and `http://cukes.info/`.

more of these executions as desirable. An example of this behaviour is where testing random position replacement produces a different position, only to have an identical position randomly generated and the inequality fail.

## 5.3 Cost Functions and Fitness

This section focuses on the production of cost values for individual arrangements within a population. This assesses how good each individual is in isolation according to some heuristics. These values are then used to produce fitness values relative to the rest of the population.

A PD for the population is produced using Goldberg's roulette wheel metaphor [Gol89]. In Goldberg's work explicit fitness values are known. These can be obtained by inverting the costs of an individual to find its fitness value. Fitness values can be any $n \in \mathbb{N}$. A consequence of this is the possibility of division by zero, so the denominator is offset to avoid this. This does not affect the ordering and prevents the necessity of an undesirable range restriction on cost functions, i.e. a solution should be able to have 0 cost.

For a population $\alpha$ of $i$ (individuals):

$$c(i) = \text{cost of individual } i \qquad \text{(Cost function)}$$

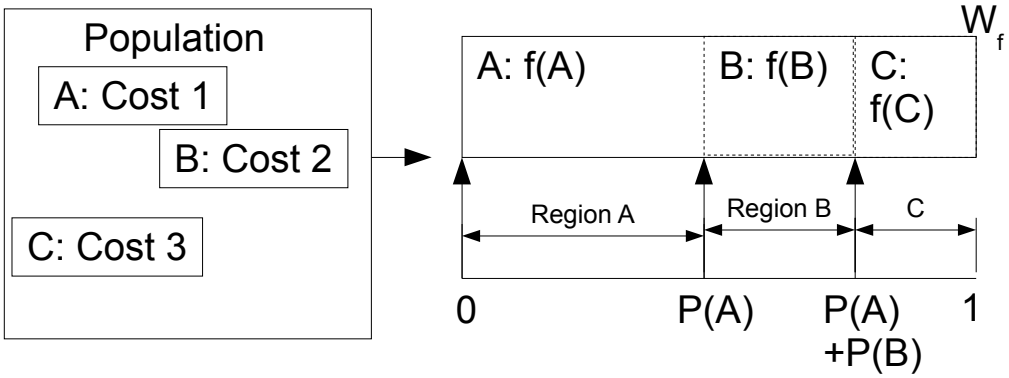
$$f(i) = \frac{1}{c(i)+1} \qquad \text{(Fitness of } i\text{)}$$

$$W_f = \sum_{i \in \alpha} f(i) \qquad \text{(Total population fitness)}$$

$$P(i) = \frac{f(i)}{W_f} \text{ such that } \sum_{i \in \alpha} P(i) = 1 \qquad (5.2)$$

It may seem that this use of the term fitness is incorrect according to [Whi94], since here fitness is not determined with respect to other members of the current population, but the fitness value definition may be rewritten as $f(i) = \frac{\sum_{i \in \alpha} P(i)}{c(i)+1}$. This is correct according to the law of alternatives (or total probability).

Figure 5.2: A Population's roulette 'wheel' example




The implementation can be visualised as a 'tape', or linear scale, representing the cumulative probability function (CPF). Each region on the tape is assigned to an individual i, where the width of the region is equal to $P(i)$. To select an individual a random number $0 \leq n \leq 1$ is generated and found on the tape. To find the individual associated with this region move

along the tape towards 0 until hitting a region boundary value (or 0). Read off the individual associated with this boundary as the result. This is illustrated in Fig 5.2.

This is implemented in Java using a sorted map and the submap tail function, where $0 \leq cpd \leq 1$.

```
Arrangement individual = wheel.get(wheel.tailMap(cpd).firstKey());
```

Greater efficiency may be obtained using lower level collections and sorting, but this implementation was deemed sufficient for experimentation. It was also determined that rounding errors will prevent the CPF from summing to 1 in most cases, but as the selection algorithm works from low to high this will not break the algorithm, rather it will result in approximation of the desired PD. The rounding error is inconsequentially small for this GA unless the population size is very high.

It is important to note that there is no notion of scaling or proportionality between these cost functions. Each cost function's results should be compared only with their own results for other individuals. This is sufficient for a GA, as it will enable producing a 'fitness landscape' as described by Mitchell [Mit99]. A consequence of this is that in evaluating a population the same fitness function must be used to assess the viability of each individual. It is certainly possible to use different cost functions in different runs, this will be explored later in evaluating the different cost functions. It may also be possible to use different fitness functions on different populations during a run, but this is not considered.

### 5.3.1 Monophonic Fret Gap Cost

When first producing the GA implementation a simple cost function was developed which ignored the LHFD and simply allocated costs based on the sum of fret distances between adjacent notes in the arrangement. This is a simple solution to the sheet music to tablature transcription problem which assumes that the solution requiring the least movement between notes will be the best. It ignores movement from string to string and does not understand the concept of open strings (which should cost little or nothing to play). Section 5.3.2 describes a function which builds on this idea to also consider LHFD.

### 5.3.2 Simple Hand Position Model

An arrangement is an ordered collection of notes being played at *positions* by associated *fingers*. An important concept in costing a sequence of positions is the LHP, which restricts the positions fingers are able to reach at a given time. The LHP is taken to represent the fret position of the left-hand's first finger.

A simple model for tracking the LHP is to produce a block of allowed finger positions for each LHP and fit this across the arrangement. This may be seen in Table 5.1 where the LHP is indicated by $f$. The first note and finger encountered will allocate a LHP and each subsequent note will either fit the LHP model, in which case the position is retained, or change the LHP to one accommodating the finger and position to be played. In this model there is no cost associated to a finger playing a fret: it is considered to be resting in that position and able to move into position easily.

The cost could be simplified to count LHP changes rather than summing the distance travelled during changes. This does not seem intuitive, so has been skipped as a starting point for the model.

The cost accumulation progressing from note to note is tracked by $\delta$, where $\delta_n$ is the distance magnitude metric between adjacent notes defined by Equations 5.3, 5.4.

Table 5.1: Left-hand Position Model Across 4 Frets

| Fret | $f$ | $f+1$ | $f+2$ | $f+3$ |
|---|---|---|---|---|
| Finger | 1 | 2 | 3 | 4 |
| Cost | 0 | 0 | 0 | 0 |

Table 5.2: Two LHP Run Examples

(a) 4 notes at 3 LHPs with associated $\delta$s

| Note | $n_0$ | $n_1$ | $n_2$ | $n_3$ |
|---|---|---|---|---|
| Fret | 5 | 7 | 3 | 10 |
| Finger | 1 | 3 | 3 | 2 |
| LHP | 5 | 5 | 1 | 9 |
| $\delta_n$ | 0 | 0 | 4 | 8 |
| $\sum \delta$ | 12 | | | |

(b) Negative LHP and Open String

| Note | $n_0$ | $n_1$ | $n_2$ | $n_3$ |
|---|---|---|---|---|
| Fret | 3 | 7 | 0 | 1 |
| Finger | 4 | 3 | 0 | 3 |
| LHP | 0 | 5 | - | -1 |
| $\delta_n$ | 0 | 5 | - | 6 |
| $\sum \delta$ | 11 | | | |

$$\delta_0 = 0 \tag{5.3}$$

$$\delta_n = |LHP(n_y) - LHP(n_{y-1})| \tag{5.4}$$

An example run is shown in Table 5.2(a).

Frets beyond the range of the instrument are implied when dealing with the top and bottom of the fret range. These are valid LHPs, though are unlikely to perform well during costing due to the reduced number of positions they offer. This is illustrated by Table 5.2(b).

An important pitfall to avoid is the treatment of open strings. A simple approach is taken which skips open strings in a sequence, ignoring their LHP and costing the notes either side of them as if the open string note were absent. This may be seen in Table 5.2(b) and a solution is presented in the code sample Figure 5.3.

### Simplifications, Flaws and Experimental Observations

A notable flaw in this model is how it treats open strings: when playing the $0^{th}$ fret no finger is required and no hand position change is required. The LHP does not change to play the $0^{th}$ fret. This breaks the adjacent costing sequence, i.e. in a sequence with frets 3, 0, 12 this model will return a cost of 9. A better model would consider the open string at note $n_1$ to provide time to move from $LHP(n_0)$ to $LHP(n_2)$, and scale down the transition cost, 9, by some ratio coefficient.

Strings are omitted from the model entirely. This simplification seems likely to hinder the results obtained, but may be sufficient for monophony. It is predicted that this model will perform well for playing scales but will fail to produce solutions in agreement with source materials and human experts when considering pieces with pauses, and the fingering of broken chords, which would see different fingers in the same fret as in the A chord in Figure 1.1.

Also absent from this model is the notions of costing changes by the time allowed for the LHP transitions.

During implementation and testing it was noted that introducing open strings resulted in significantly more variation appearing in the low cost solutions being produced for the C major scale problem. This resulted from open strings at strategic points allowing the overall LHP to remain the same while producing significant variations in fingering and open string use. While

Figure 5.3: Java Code Example: LHP Cost Allocation

```
public static void assign_simple_hand_model_cost(Arrangement arr){
  Iterator<FingeredNote> itr = arr.iterator();
  FingeredNote note = itr.next();
  FingeredNote previous_note = note;
  int delta_sum = 0;

  while (itr.hasNext()){ //the first evaluation is 1 against 2
    note = itr.next();
    int lhp_this_n = lhp_of_fingered_fret(note.fret(), note.finger());
    int lhp_previous_n = lhp_of_fingered_fret(previous_note.fret()
                          , previous_note.finger());
    if(note.fret()==0){
      // don't cost 0 fret and don't advance previous note
      //cost across the open string as though it isn't there
    }else{
      delta_sum += Math.abs(lhp_this_n - lhp_previous_n);
      previous_note = note;
    }
  }
  arr.assign_cost(delta_sum);
}
```

these are valid under this cost scheme, and may be beneficial for playing real music, for scale playing this was a bad result. It broke the desired pattern of progressing through the scale 1 string at a time into skipping between strings as the scale progressed, as in [Win02]. Example solutions from the system are presented in Table 5.3, where solution 2 is expected, solution 3 is an acceptable solution utilising open strings, and solution 1 is an undesirable solution with skipping between strings. For a guitar string order reference see Figures 1.1, 2.5.

Introspectively this was noted as making the solutions considerably more cognitively and physically difficult to remember and play. It was decided that a string change cost feature is desirable to model this. While the paper thus far has restricted its focus to the left hand it is believed that this shows the importance of the right hand also in determining costs. It does not expand the search space any further as we are not considering different options for how the right hand may strike a note, only that (for monophony) it prefers playing the same string to skipping between strings a distance apart.

### 5.3.3 Hand Position Model With Stretching

A modification of the model presented in Section 5.3.2 includes stretching the first and fourth fingers of the left hand to reach more difficult, but possible, fret positions. These are more desirable than a change of hand position in some cases, depending on surrounding context. Stretching will have a cost associated with it, as it is more difficult than remaining within the ordinary hand model range of 4 frets. This cost should be proportional to the cost of LHP movement, with a weighting coefficient tuneable to reach the desired effect. This lends itself well to tuning by machine learning approach, such as that used in the papers by RD and others

Table 5.3: Cost 0 Solutions for C Major Scale Showing Open String Issue In Scale Playing

| | Solution 1 | | | Solution 2 | | | Solution 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| Note | Finger | Fret | String | Finger | Fret | String | Finger | Fret | String |
| C1 | 4 | 8 | LOW E | 2 | 3 | A | 3 | 3 | A |
| D1 | 0 | 0 | D | 4 | 5 | A | 0 | 0 | D |
| E2 | 3 | 7 | A | 1 | 2 | D | 2 | 2 | D |
| F2 | 4 | 8 | A | 2 | 3 | D | 3 | 3 | D |
| G2 | 0 | 0 | G | 4 | 5 | D | 0 | 0 | G |
| A2 | 3 | 7 | D | 1 | 2 | G | 2 | 2 | G |
| B2 | 0 | 0 | B | 3 | 4 | G | 4 | 4 | G |
| C2 | 1 | 5 | G | 4 | 5 | G | 1 | 1 | B |

Table 5.4: Left-hand position model extended to 6 frets

| Fret | n-1 | n | n+1 | n+2 | n+3 | n+4 |
|---|---|---|---|---|---|---|
| Finger | 1 | 1 | 2 | 3 | 4 | 4 |
| Cost | stretch | 0 | 0 | 0 | 0 | stretch |

in Chapter 3.

A further modification could consider finger positions which do not match the current LHP exactly, but are close enough to remain in the same LHP and allocate an additional cost derived from the Euclidean distance (frets) from expected to actual finger position.

### 5.3.4 Hand position model with transition timing

This model aims to extend 5.3.3 to cost LHP transitions by the time they have to occur. Complexity arises in traversing open string positions, but may be handled in a recursive fashion. A model making use of timing information is essential in moving up from scale playing, where timing is somewhat arbitrary, to playing musical pieces as intended where note pitch, duration and rests are all essential to the intent of the piece.

Two options are suggested, a simplified model where beat length is used, and a correct system where real-time note length is used. The latter requires users to input the tempo at which a piece will be performed. The simplified system may be retrofitted with a beat length coefficient to act in real time.

Completion and implementation of this cost function are left as further work (Section 7.2).

### 5.3.5 String Change Cost Model

Two trivial approaches present themselves for reducing string changing during a piece. One counts the number of changes, and the other tracks the total number of strings traversed during play, for example Table 5.5. The model chosen for implementation was the one counting the total number of strings traversed.

### 5.3.6 Heijink Neck Position Preference Model

Derived from the work of HM [HM02] two neck positions are found to be desirable. It was predicted that medium LHPs would be preferred, and found experimentally that low frets were

Table 5.5: String Change Costing

| String | Changes Cost | Traversal Cost |
|---|---|---|
| A | 0 | 0 |
| A | 0 | 0 |
| D | 1 | 1 |
| B | 1 | 2 |
| A | 1 | 3 |
| Total | 3 | 6 |

used most. Introspectively it is believed that a medium position is preferable, though a decision subjective to the piece and individual. Three models are presented here: low positions preferred, medium neck positions preferred, and high neck positions discouraged.

A simple way to cost low hand positions as preferable, which works in isolation, is to sum the fret values for each note in a solution. This is crude, but will obtain similar results to LHP or other abstractions.

$$\sum fret(\textit{fingered note}) \tag{5.5}$$

Medium positions may be made beneficial by a slight modification to the low position algorithm, where a central fret position is selected and the fret distance from this is summed across an arrangement.

$$\sum |\textit{central fret} - fret(\textit{fingered note})| \tag{5.6}$$

High positions may be discouraged (roughly) by taking the low hand position algorithm and raising each term in the summation to an arbitrary power, for this implementation it was chosen to square the terms. This final solution is deemed preferable.

$$\sum fret^2(\textit{fingered note}) \tag{5.7}$$

## 5.4 Composite Cost Functions and Coefficient Tuning

A combined model can be constructed from other cost functions by weighted summation. Consideration should be given to avoid solutions being penalised several times for the same flaw in the different cost functions, i.e. attempts should be made to make the components disjoint, or orthogonal. An example composite cost function is

$$cost(i) = \alpha cost_{string}(i) + \beta cost_{LHP}(i) + \gamma cost_{fingering}(i)$$

where $i$ is an individual and $\alpha, \beta, \gamma$ are weighting coefficients.

HM [HM02] conclude that there are 3 factors affecting a classical guitarist's choice of fingering sequences: hand position on the guitar neck, required finger span, and hand repositioning within note sequences. Following HM directly would produce a cost function of the form

$$cost(i) = \alpha cost_{position}(i) + \beta cost_{span}(i) + \gamma cost_{repositioning}(i)$$

A simple implementation of this for monophony was produced using the cost functions described in 5.3.4 and 5.3.6, where 5.3.4 handles repositioning and span (stretching).

Table 5.6: Cost Value Weighting Effect On Fitness

| Cost | Fitness ($cost \times 1$) | Fitness ($cost \times 2$) | Fitness ($cost \times 10$) |
|---|---|---|---|
| 0 | 0.464 | 0.587 | 0.851 |
| 1 | 0.232 | 0.196 | 0.077 |
| 2 | 0.155 | 0.117 | 0.0405 |
| 5 | 0.0774 | 0.053 | 0.0167 |
| 10 | 0.0422 | 0.0279 | 0.00843 |
| 15 | 0.029 | 0.0189 | 0.00564 |

### 5.4.1 Effect of Weighting Costs on Fitness

Multiplying cost values by coefficients as described will also effect the fitness values produced by each cost function, and may also be useful for tuning cost functions used in isolation. A high multiplication coefficient will see low cost individuals become significantly fitter than high cost individuals, more so as the coefficient increases in size. Taking for example the arbitrary LHP costs seen in Table 5.6 it is seen that by introducing weighting rather than simply affecting the proportionality of one cost function's significance against other cost functions the cost function's behaviour in isolation is also affected significantly.

An interesting follow-up from this would be to consider non-linear modifications of cost values prior to returning them, such as squaring them. As no strict evaluation function is being followed (due to its absence in the literature) any such choices currently would be arbitrary.

### 5.4.2 Machine Learning Approaches

Approaches were seen in the literature survey to address the issue of cost function tuning which are very relevant to this study. Given time constraints manual tuning has been used to optimise the cost functions, but an approach similar to that used by TP (GAs optimising GAs and cost function parameters) or RD (Section 3.2.2) would be preferable and should be considered in further work.

## 5.5 Cost Function Shortcomings

A major facet of the literature survey was the analysis of the Adjacent Note Assumption. These cost functions have not made use of this additional information. This is reiterated in the further work proposals, Section 7.2.

# Chapter 6

# Results and Discussion

As previously discussed by TP in Section 3.2.3 this problem does not lend itself well to numerical evaluation due to the diversity and subjectivity of good solutions. It was intended to produce a user study of this system, but given its state of development there was little point as considerable work is required before it reaches a state mature enough for acceptance testing. Instead experimentation here pertains to the current capabilities of the algorithm, the effects of its parameters, and consideration of the segmentation necessity. Considerable scope is left for further work, discussed in Chapter 8.

## 6.1 Cost Function Observation

This section is intended to observe results produced by the composite cost function described in 5.4. Results will be observed with parts of the function disabled by zero coefficients, and in a small number of varied coefficients. The intent of this experiment is to illustrate properties of solutions deemed good and their correlation with the intents of the constituent cost function components. Given the arbitrary scaling of cost function values it may be of interest to look ahead to the graphs of cost value trends by generation in Section 6.2 to get an idea of how close a solution may be to the optimum for the cost function and this particular problem.

Suggestions are made in the following for adjustments to solutions which seem beneficial, but this does not necessarily correspond with what the cost function, for its parameters at that time, believes makes for a good solution. This is not easy to address, since this issue of function approximation is the crux of the field of machine learning [Mit97]. Rather the intent is to show some real results for the system in a less abstract, more hands-on manner than elsewhere in this report.

These solutions were taken from the best-result set of a 30k population 100 generation run for the C major scale (Figure E.1) using the composite cost function described in Section 5.4 weighted (neck position, string change, LHP movement) : (1, 30, 60).

As a reminder of the structure this block solution represents a GA genotype, or in this context a phenotype since it is being considered to play on a guitar. Each row of text is a chromosome, of the form

```
{chromosome note invariants} at gene; gene; gene
{[note]|start beat|duration} at finger; fret; string
```

Figure 6.1 was the best solution observed during the run. It should be noted that this solution is not ideal in that the string ordering is inconsistent with typical scale playing, the

Figure 6.1: A solution for the C Major Scale

```
{[C2]|st:0.0|dr:1.0} at fi4;fr10; str:D
{[D2]|st:1.0|dr:1.0} at fi1;fr7; str:G
{[E3]|st:2.0|dr:1.0} at fi0;fr0; str:HIGH_E
{[F3]|st:3.0|dr:1.0} at fi2;fr6; str:B
{[G3]|st:4.0|dr:1.0} at fi4;fr8; str:B
{[A3]|st:5.0|dr:1.0} at fi1;fr5; str:HIGH_E
{[B3]|st:6.0|dr:1.0} at fi3;fr7; str:HIGH_E
{[C3]|st:7.0|dr:1.0} at fi4;fr8; str:HIGH_E
Cost: 321
Generation: 46
```

Figure 6.2: A solution for the C Major Scale

```
{[C2]|st:0.0|dr:1.0} at fi4;fr10; str:D
{[D2]|st:1.0|dr:1.0} at fi1;fr7; str:G
{[E3]|st:2.0|dr:1.0} at fi3;fr9; str:G
{[F3]|st:3.0|dr:1.0} at fi4;fr10; str:G
{[G3]|st:4.0|dr:1.0} at fi4;fr12; str:G
{[A3]|st:5.0|dr:1.0} at fi2;fr10; str:B
{[B3]|st:6.0|dr:1.0} at fi4;fr12; str:B
{[C3]|st:7.0|dr:1.0} at fi4;fr13; str:B
Cost: 323
Generation: 80
```

Figure 6.3: A solution for the C Major Scale

```
{[C2]|st:0.0|dr:1.0} at fi4;fr10; str:D
{[D2]|st:1.0|dr:1.0} at fi4;fr12; str:D
{[E3]|st:2.0|dr:1.0} at fi1;fr9; str:G
{[F3]|st:3.0|dr:1.0} at fi1;fr10; str:G
{[G3]|st:4.0|dr:1.0} at fi2;fr12; str:G
{[A3]|st:5.0|dr:1.0} at fi3;fr14; str:G
{[B3]|st:6.0|dr:1.0} at fi1;fr12; str:B
{[C3]|st:7.0|dr:1.0} at fi2;fr13; str:B
Cost: 456
Generation: 90
```

Figure 6.4: A solution for the C Major Scale

```
{[C2]|st:0.0|dr:1.0} at fi2;fr10; str:D
{[D2]|st:1.0|dr:1.0} at fi4;fr12; str:D
{[E3]|st:2.0|dr:1.0} at fi0;fr0; str:HIGH_E
{[F3]|st:3.0|dr:1.0} at fi1;fr10; str:G
{[G3]|st:4.0|dr:1.0} at fi3;fr12; str:G
{[A3]|st:5.0|dr:1.0} at fi1;fr10; str:B
{[B3]|st:6.0|dr:1.0} at fi3;fr12; str:B
{[C3]|st:7.0|dr:1.0} at fi4;fr13; str:B
Cost: 323
Generation: 275
----
```

following two higher costed solutions do not portray this property. In a longer run either a better solution would arise, or the cost function coefficients are incorrect to give the desired results.

Figure 6.2 is consistent with string use in scale playing but involves two LHP changes and possible but (subjectively) uncomfortable use of the 4th finger towards the end.

Figure 6.3 demonstrates a good candidate for improvement. It has the same starting point as the previous solution but progress with an early LHP change (incurring cost). In the latter half it uses LHP changes to make use of fingers 1 and 2 where it could instead use fingers 3 and 4 and accrue less cost.

Figures 6.4 and **??** show the two best observed solutions from a longer run (400 generations) of the same problem. An error still exists here, in that note 3 involves a string change to an open string where it would be preferable to play string G at the same LHP, i.e. replacing with

```
{[E3]|st:2.0|dr:1.0} at fi1;fr9; str:G
```

and then continuing with the same LHP for the subsequent notes as far as possible.

Figure 6.5 shows the best solution found during a 400 generation run with cost parameters (neck position, string change, LHP movement) : (1, 60, 60). It can be immediately observed that the strings used are consistent with those expected for scale playing. Indeed this was

Figure 6.5: A solution for the C Major Scale For Different Cost Function Parameters

```
{[C2]|st:0.0|dr:1.0} at fi1;fr1; str:B
{[D2]|st:1.0|dr:1.0} at fi3;fr3; str:B
{[E3]|st:2.0|dr:1.0} at fi3;fr5; str:B
{[F3]|st:3.0|dr:1.0} at fi4;fr6; str:B
{[G3]|st:4.0|dr:1.0} at fi4;fr8; str:B
{[A3]|st:5.0|dr:1.0} at fi1;fr5; str:HIGH_E
{[B3]|st:6.0|dr:1.0} at fi3;fr7; str:HIGH_E
{[C3]|st:7.0|dr:1.0} at fi4;fr8; str:HIGH_E
Cost: 347
Generation: 84
```

consistent across the top 8 results observed for that particular run. This demonstrates the effect of changing cost function parameters on the obtained results. There is a lot of room for further work on this issue, and to address it properly manual tuning must be left behind and a machine learning approach to function approximation taken to optimising the cost function parameters for a range of pieces. Further work is proposed in Section 7.13.

## 6.2 GA Experimentation

GA parameters have been investigated extensively elsewhere and are discussed in [Gol89, Whi94, Mit99]. However, given the immaturity of the system it was decided to empirically investigate some design decisions and also demonstrate its functionality as a GA.

M. Mitchell [Mit99] mentions GA researchers often reporting statistics, such as the best fitness found in a run and its generation of discovery, averaged over repeated runs for the same problem. These results will be different because of the probabilistic methods employed. The system presented is easily modified to produce data for analysis with MATLAB or similar tools. Code was added to the system which for each run produced a cost value digest such that each generation's cost values are printed as a space delimited row. This file is then read into MATLAB using the `load` function and basic statistical analysis performed and plotted. These files varied in size from tens of megabytes to gigabytes depending on the population size and run length used.

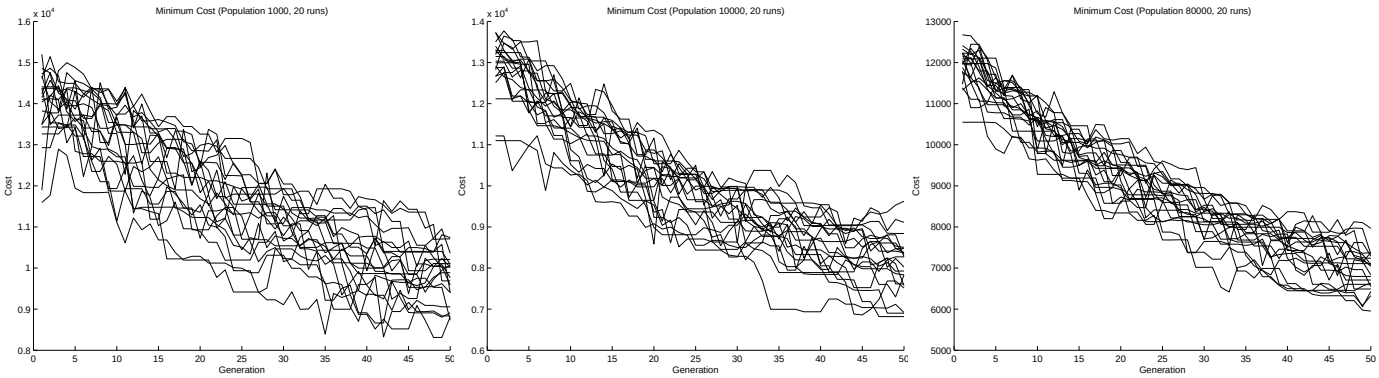
It should be noted that the range of cost values are essentially arbitrary, determined by the cost functions and their weightings. A solution with a cost of 0 according to one function may have a cost of 300 according to another function. Evaluation of solutions must be made in another fashion, and is discussed in Section 6.1. The presented graphs in this section are useful in visualising GA behaviour only, not in the quality of the results produced.

Figures 6.7, 6.8,6.9,6.6 show examples of GA runs over a piece of real music (Figure E.2). From these several properties of the GA can be observed.

Figure 6.6 demonstrates the optimisation capability of a GA directly, by showing that as the run progresses better solutions are found, in this case by minimising a cost value. As previously mentioned a GA search is not guaranteed to find a globally optimal solution, but will approach it in a guided stochastic manner as shown in the plots.

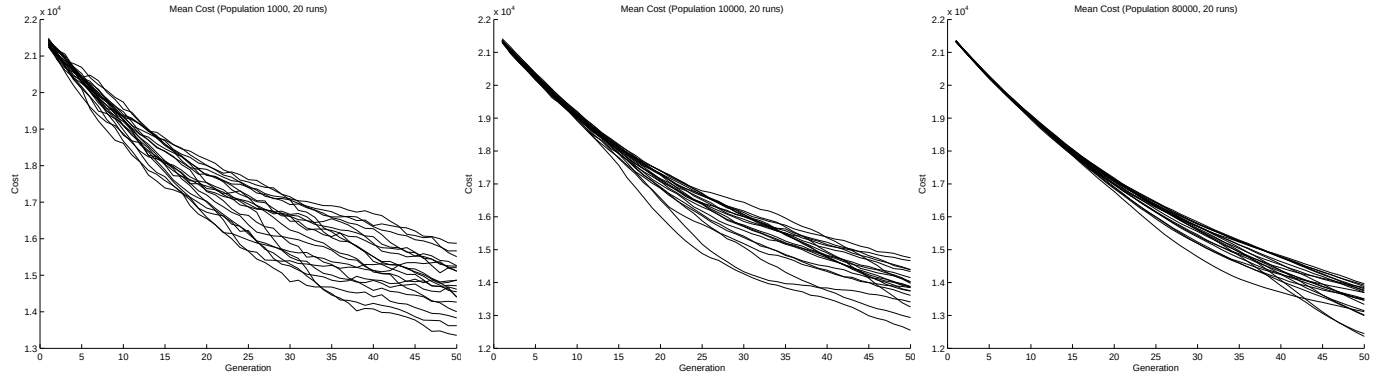
Figure 6.7 shows the mean cost value for the generations reducing with time. This indicates that rather than individuals randomly happening to have good solutions, effectively being a

Figure 6.6: Sailor's Hornpipe: Varied Populations Size Minimum Costs




(a) Population 1k Minima (b) Population 10k Minima (c) Population 80k Minima

Figure 6.7: Sailor's Hornpipe: Varied Populations Size Means




(a) Population 1k Means (b) Population 10k Means (c) Population 80k Means

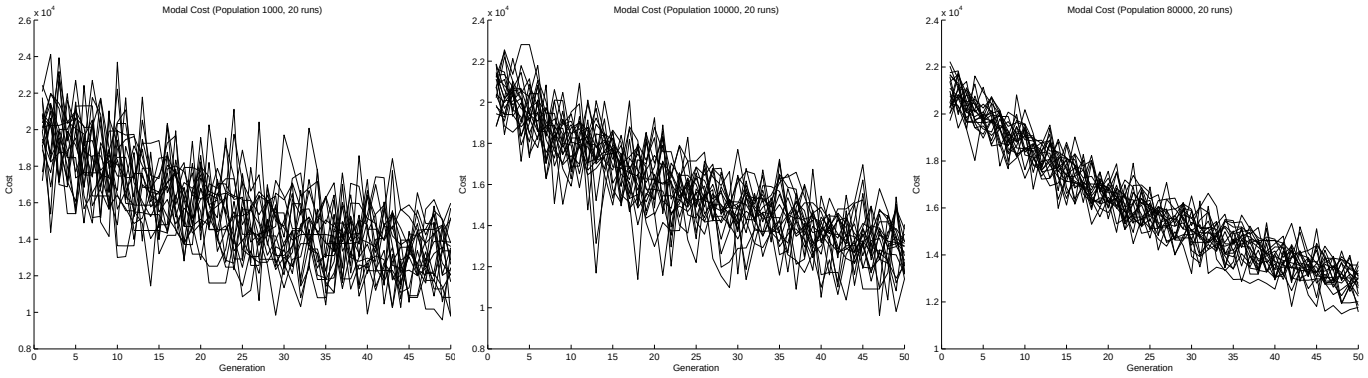
random search, the population as a whole is improving as the run goes on. This demonstrates the natural selection concept that GAs are based upon.

An important observation in these graphs, in particular of the median and modal values, is that with larger population sizes the plots appear more uniform or regular. This is an example of the central limit theorem, the observation that given sufficient examples a random process will tend towards a central value. In practical terms for the GA this means that it will be more reliable with large population sizes, for example in the minima plots (although the axes are scaled differently) the 80k population more reliably reaches fitness 6000-8000 than the 1k and 10k populations. The general performance and reliability of the system is important in considering moving from an experimental to practical situation, as a user will want the system to always produce good results.

### 6.2.1 Run Length: Visualising Fitness & Determining When to Halt

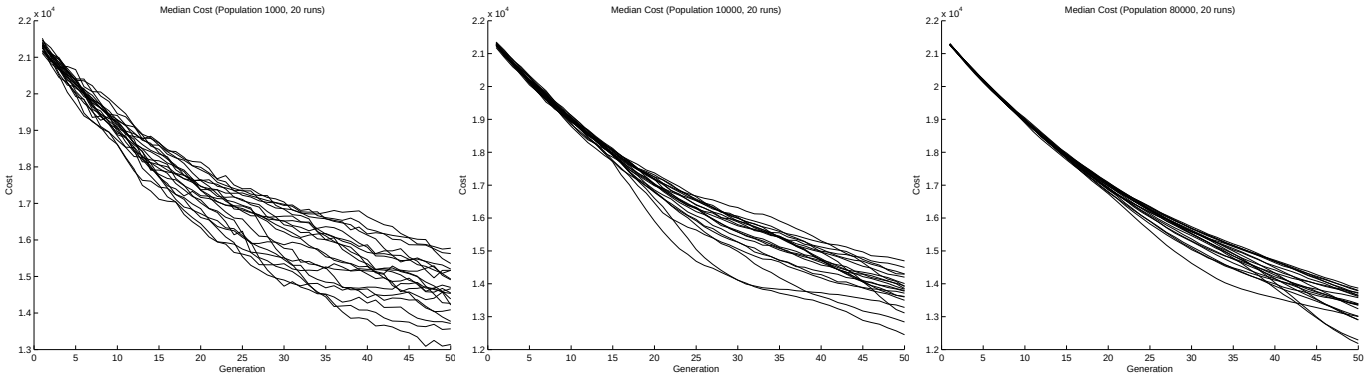
An issue with the system presented thus far is the notion of a fixed run length. When starting a run, without prior experience of the given problem and GA parameters in use it will be difficult to judge what run length to use. If it is too short minimum cost solutions will not be converged on, too long and they will be found but at the cost of wasted processing time. This may not seen like an issue at first glance, but during experimentation it was found that for a monophonic piece

Figure 6.8: Sailor's Hornpipe: Varied Populations Size Modes



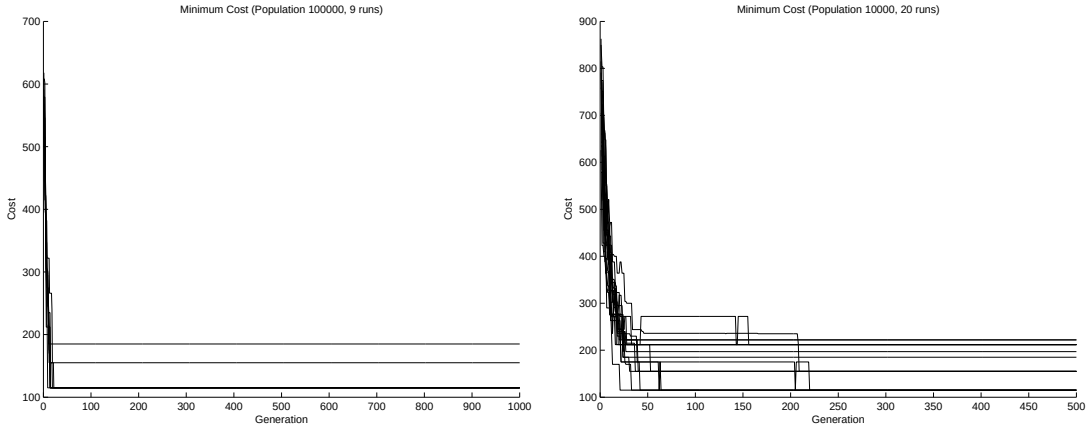

(a) Population 1k Modes (b) Population 10k Modes (c) Population 80k Modes

Figure 6.9: Sailor's Hornpipe: Varied Populations Size Medians




(a) Population 1k Medians (b) Population 10k Medians (c) Population 80k Medians

Figure 6.10: Extended Length C Major Scale Minimum Cost Values




(a) 100k Population, 1k Generations, 9 Repeats (b) 10k Population, 500 Generations, 20 Repeats

Figure 6.11: Sailor's Hornpipe Long Run

approaching 50 notes, using approaching 1GB of memory, after 24 hours a good solution had not yet been found. In this situation the chosen stopping generation is a somewhat arbitrary and a better solution should be found. A proposal to address this issue is made in Section 7.6 Following is a graphical representation of some GA run statistics and a simple proposal for addressing the halting issue with the current system design.

A graphical interpretation of this problem follows. Two problems from Appendix E are considered, a simple scale playing problem and a 50 note verse from the Sailor's Hornpipe (Figure E.2), with varied population sizes (time and memory) and run lengths (time).

### Scale Playing

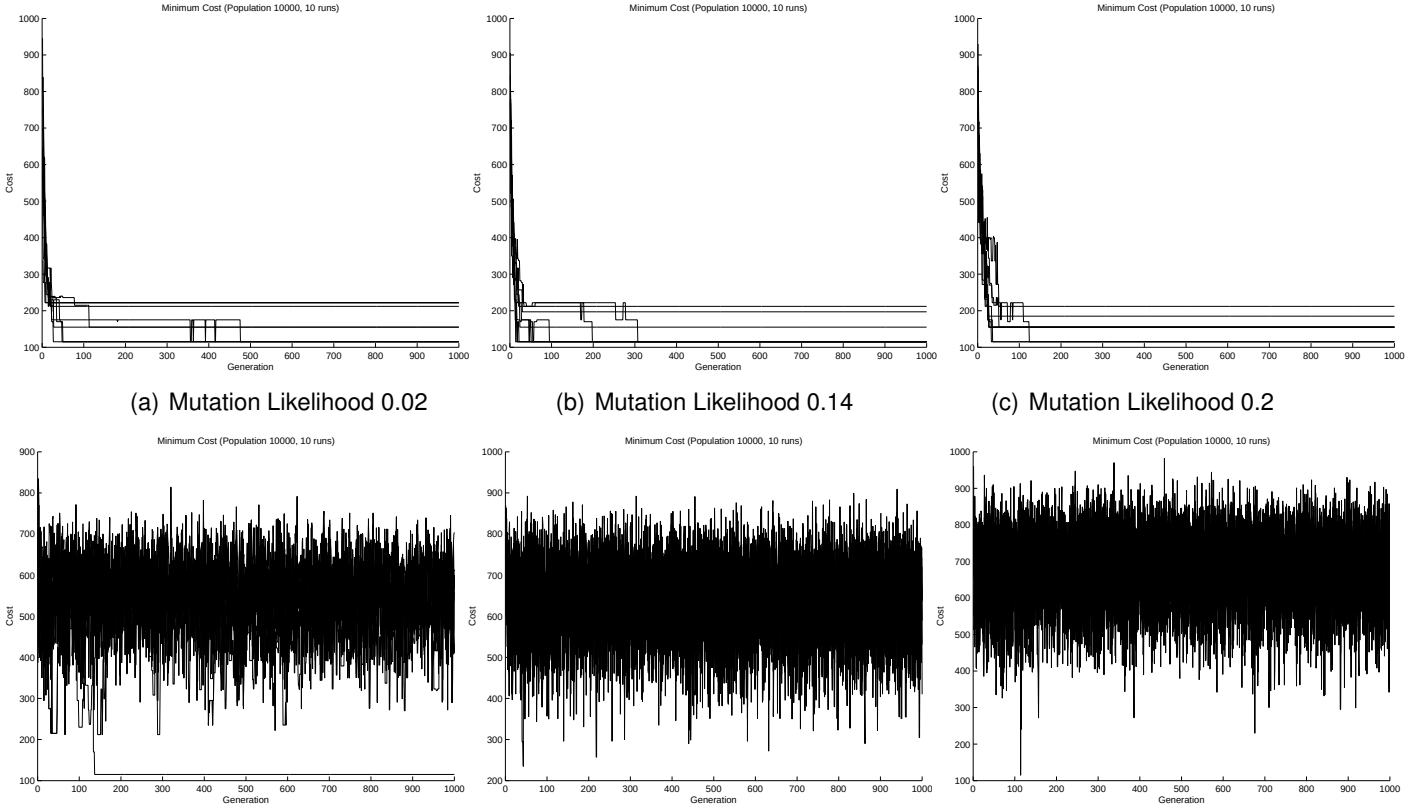
The C Major scale presented in Figure E.1 is an 8 notes monophonic score without any importance placed in the timing information. It and similar length problems have proven quite tractable with the developed system. Two plots, found in Figure 6.10, show the minimum cost present in each generation during a run. The plots show several repeated runs with the same parameters on the same axes. Visualising these results will aid in understanding the effect of halting at different times. The Figures in 6.10 show the populations means converging on low costs as the run proceeds, at around generation 50 for this example, but optimal (low) costs appear to find and settle on local rather than global optimums. This should be corrected by mutation during a sufficiently long run, and some evidence of this may be visible in Figure 6.10. The effect of varying mutation on observed results is investigated in Section 6.2.2.

### Segment Processing

The Sailor's Hornpipe segment presented in Figure E.2 is considered a reasonable (minimum) user input for the system. The minima for its execution are seen in Figure 6.11.

As an aside although presented here as an entire piece, it is actually one of three segments, where the third is a repetition with an alternative ending. The piece was manually segmented or shortened because of the difficulty of entering data rather than for computational tractability. A proposal for further work in Section 7.11 presents a solution to this score input issue.

Figure 6.12: C Major Scale: Varied Mutation Likelihood Minimum Costs




(a) Mutation Likelihood 0.02 (b) Mutation Likelihood 0.14 (c) Mutation Likelihood 0.2

(d) Mutation Likelihood 0.3 (e) Mutation Likelihood 0.4 (f) Mutation Likelihood 0.5

### 6.2.2 Varying Mutation

As seen in runs appear to stagnate after a number of generations, despite reaching divergent optimums. While it is true that the problem domain has divergent and structurally incompatible minima it should be possible, through mutation, for a population to discover other maxima. This robustness against local maxima is one of the promoted advantages of the GA. This experiment observes the affect of the chromosome mutation likelihood on population development.

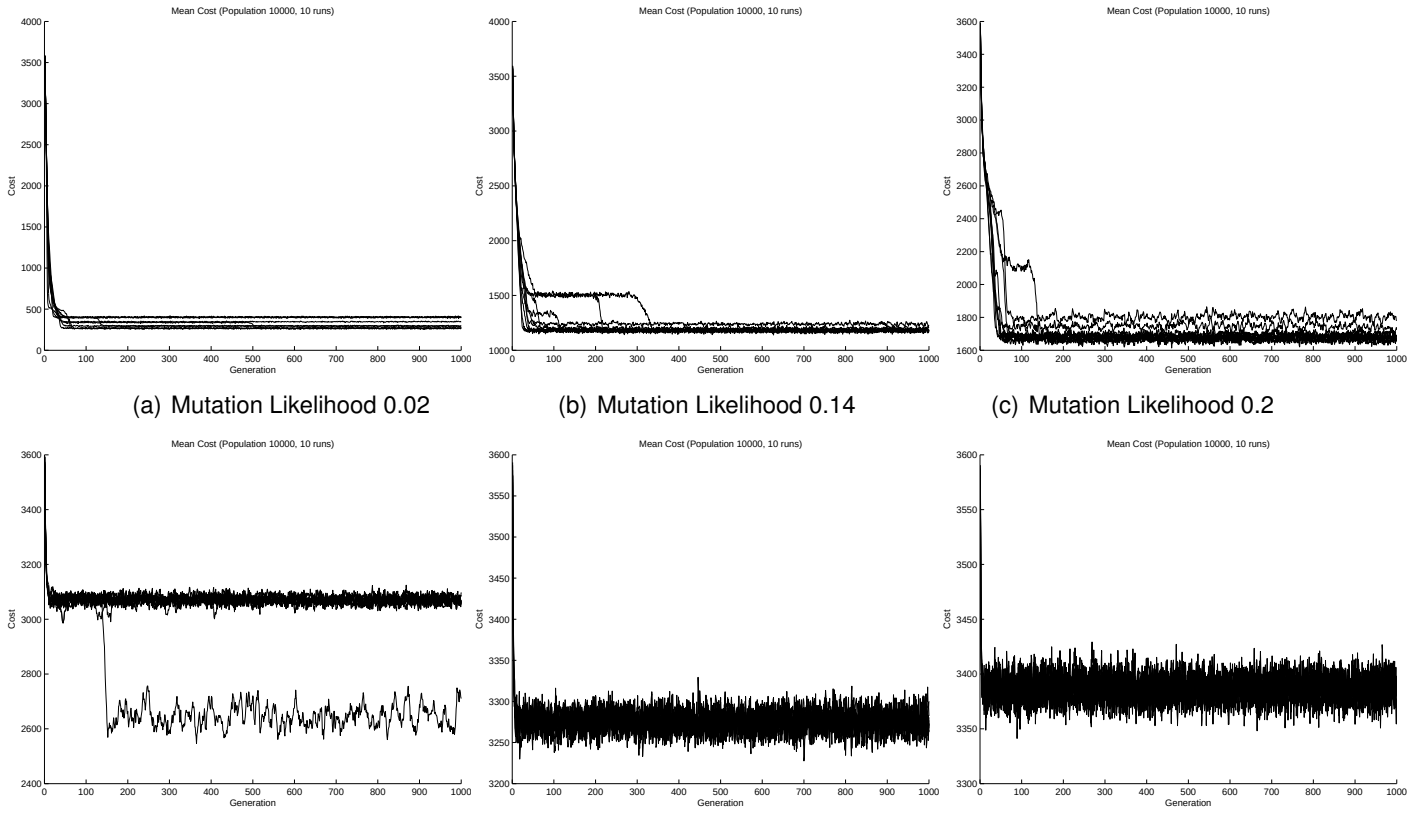
Figures 6.13, 6.12 clearly show that there is an optimal setting for mutation between it being too low to reinvigorate stagnant populations and being so high that it turns the GA into a purely random process unable to guide itself due to excessive randomisation of offspring negating the positive effects of selection and recombination in the system. The figures observed suggest that an optimal value will be found between 0.2 and 0.3, though whether this applies to other musical scores also should be investigated further (proposal 7.4).

### 6.2.3 Varying Crossover

Crossover is considered an important parameter as it will determine how good solutions are recombined in the hope of producing better solutions. If the strategy is ineffective the algorithm will still work, as poor solutions will be discarded, but more individuals will need to be processed, by increasing the population size or length of the run, to reach good solutions.

Two strategies will be considered with varying parameters for the C major scale task. Random locus crossover will stochastically determine which locus to perform crossover recombi-

Figure 6.13: C Major Scale: Varied Mutation Likelihoods Means




(a) Mutation Likelihood 0.02 (b) Mutation Likelihood 0.14 (c) Mutation Likelihood 0.2

(d) Mutation Likelihood 0.3 (e) Mutation Likelihood 0.4 (f) Mutation Likelihood 0.5

nation about. This will be considered with low, high and proportional to genotype length likelihoods. Midpoint crossover will split about the central locus, if crossover is to be performed. Midpoint crossover will be tested for three likelihood values across two orders of magnitude, ranging from likely (above half chance) to unlikely.

Success will be determined by assessing which strategy resulted in good values being found earliest in their runs, by making use of the best-result set. A decision must be made how to assess this, some strategies may quickly find a small number of optimal solutions while others may find optimal solutions more slowly but maintain a broader search and produce a larger number of good solutions. Rather than specify a target before seeing results it was decided to obtain results and assess them after-the-fact.

# Chapter 7

# Further Work

Following are several proposals for further work, either issues which arose as interesting during the project but were deemed beyond its scope or those parts of the project which were not completed due to time constraints.

## 7.1 Polyphony

An important advance for the system will be moving from monophony to asynchronous polyphony, where notes may begin independently of one another rather than being clustered into chord groups starting and ending together. This will significantly increase the search space and require redesigning of the data encoding and cost functions.

## 7.2 Cost Function Timing

Current cost functions have not made use of the stored note durations or start times. This is not correct, as notes occurring in quick succession have different constraints to those occurring after rests. A design was suggested in Section 5.3.4 but not completed or implemented.

## 7.3 Cost Function Accuracy

Work has been omitted on an assessment of the quality of solutions produced by the system with its most mature cost functions. Although desirable to add new features such as in proposals 7.1 and 7.2 it may be that the system is already producing solutions good enough for use under some constraints, such as being limited to monophonic music, having to manually segment pieces of above a certain length, and having to do some manual arrangement of output pieces to improve on their flaws. Examining this process would be helpful in identifying weaknesses of the cost functions and improving them. A machine learning approach considering a set of scores with known good results would also be a useful study, as discussed in Section 4.2.

## 7.4 GA Tuning

During experiment 6.2.2 it was seen that an optimum value exists for the mutation likelihood, and that manual tuning and graphing to approach this value is possible. It would be preferable

for this process to be automated. Using statistical techniques it should be possible to address this. A similar approach may also be taken for other GA parameters to attempt to find optimal performance without hours of manual tuning and testing.

A concern with tuning in this way is it assumes the orthogonality of GA parameters, that is that this mutation likelihood will be good irrespective of population size and crossover likelihood, and that it will be effective for all input datas. TP used a GA to optimise their GA in a machine learning fashion, which would address this orthogonality issue.

## 7.5 Representation and Costing of Embellishments and Ornamentation Techniques

Embellishments such as slides and string bending have been omitted thus far, but methods of adding them to the data encoding should be investigated. As a starting point it is thought that having data about graph edges (transitions) rather than only the vertices they connect would enable this. It is thought that an object system will lend itself better to this task, and experimental extensibility to it, than a low level encoding, showing a benefit of the chosen implementation method.

## 7.6 Run Halting Problem Proposal And Object Serialisation

It is suggested that the system execute until the best-result set is unchanged for n generations. At this time the user should be prompted with the current results and the option to halt or continue running in search of better solutions. If the user wishes to continue they should be able to continue as previously or perform a mass-mutation of the population, whereby for one generation the likelihood of mutation occurring is increased in the hope of introducing new concepts into the population. This is used at the user's discretion. These controls should be trivial to implement with modifications to the current system.

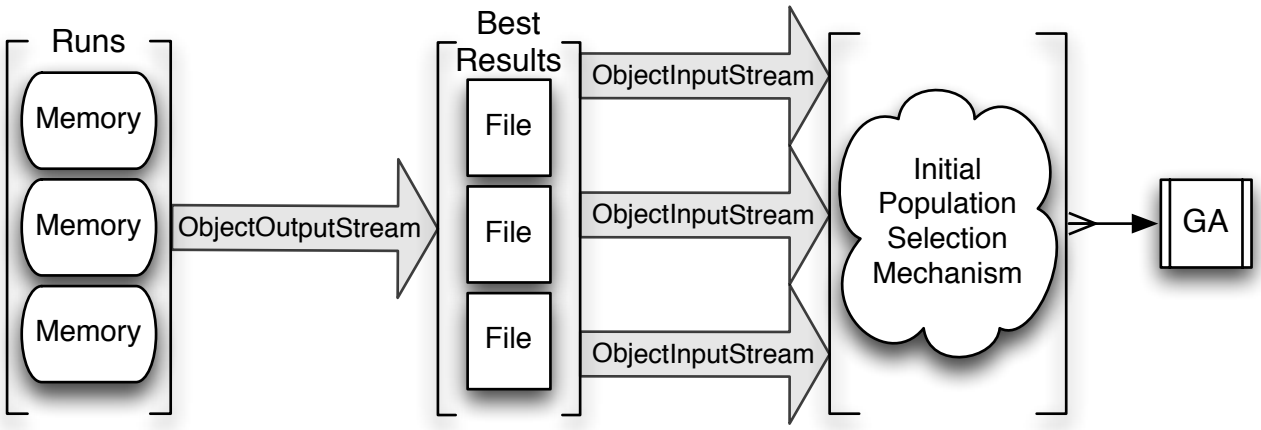
The ability to continue a run in this fashion is important getting around the run length determination issue presented in Section 6.2.1. This concept could share implementation with the distributed object transfer mechanism required for proposal 7.7 (using Java's object serialisation mechanism) in halting execution of a population, saving all of the objects, and returning to it at a later time to continue execution with those objects.

## 7.7 Distributed GA

As described by TP a distributed approach lends additional tractability to the problem and may result in better results given the increased initial diversity produced by having several initial populations evolve and later coalesce.

Rather than the islanding scheme proposed where populations run in parallel with individuals transferring between them, a scheme more amenable to atomic batch processing has been devised for this system. Figure 7.1 is a high level illustration of this system. In Java storing objects in files is trivial, they are marked as implementing an interface, a file output stream is created which writes objects, and a method is called to save the objects. All of this work is performed by the Java libraries. Reading the files back simply requires opening the file, reading objects iteratively and casting them to the known class, in this case they would be `Arrangement` objects.

Figure 7.1: A Serialised Distributed GA Processing Scheme




The selection method is less trivial as it is not desirable, or indeed possible, to store all competing solutions from previous runs in memory at once while determining fitness values. One approach for this problem is to store solutions in memory only for as long as necessary and storing a reference or record of which file and where a solution's object may be found. Another approach is to select some predetermined number of solutions from each best results set, though this assumes a homogeneous distribution of good solutions across the set of best-result sets.

## 7.8 Steady State GA

TP refer to Whitley's Genitor [WS90] in their GA design. Two aspects are mentioned, the distributed islanding approach mentioned in proposal 7.7 and also the notion of a steady state population. In a steady state population individuals are changed one at a time rather than creating distinctly partitioned generations. It would be of interest to implement this evolutionary concept as an alternative mechanism in FINGAR which could then be compared with the current generation based model.

A straight comparison would be difficult since the generation axis used in the plots herein would not exist for the steady-state approach. A rough metric could be execution time, though care should be taken with interpreting this. It may be possible to count crossover operations in both systems and plot cost reduction comparisons with this, where the generation system would perform many at once and produce a stepped plot, while the steady-state system would produce a continuous (natural numbers) plot.

## 7.9 Dynamic Programming Considerations

Given the simplicity of current cost functions it may be of interest to again consider the dynamic programming ideas seen in Section 3.2.2. Consideration should be given to switching from the Viterbi algorithm to a more general algorithm such as Dijkstra's algorithm or another graph search algorithm which does not introduce the Adjacent Note Assumption.

| Population Size | Memory Usage |
|---|---|
| 10k | 136MB |
| 30k | 378MB |
| 60k | 735MB |
| 100k | 1137MB |

(a) Before Optimisation, 48 Notes

| Population Size | Memory Usage |
|---|---|
| 10k | 136MB |
| 30k | 375MB |
| 60k | 730MB |
| 100k | 1034MB |

(b) After Optimisation, 48 Notes

Figure 7.2: Memory Optimisation Results

## 7.10 Fingering Memory Across Segments

An important issue not addressed in the literature seen is the idea of maintaining consistent finger use and chord shapes across independent segments of music, where the Phrasal Independence Assumption has been used to determine optimal solutions. This may be used in joining together the solutions of solved phrases to form an entire solution in an automatic fashion. It may be useful for the joining mechanism to be able to modify solutions to impose this consistency upon solutions which have otherwise diverged.

For this to be implemented it would seem useful to maintain a memory not of the entire solution, but rather of used chord shapes and fingering patterns. An optimal global solution could perhaps be one which has the least fingering patterns to store, and thus the least for the guitarist to learn and remember when performing a solution.

## 7.11 MusicXML File Reading

The system currently has no operator friendly method for inputting different musical scores. They must be manually encoded into the suitable data structure in Java source code ready for processing, and compiled into the Java implementation of the system. Switching between different test pieces requires changing this data structure, this has been done using static functions to create different scores, and one of these is used as desired. While this allows for quick switching between already encoded problems at compile time it is far from ideal.

A desirable usability extension for the system is the ability to take MusicXML files as input to process at runtime. This format is an available output from the GuitarPro and TuxGuitar (free open source alternative) applications. This extension would greatly relieve the burden placed on experimenters when entering different pieces of test data to evaluate cost functions and also build towards a system for non-technical users.

## 7.12 Memory Concerns

An important concern in system design is the efficiency of the data structure devised. Practical limitations existed in terms of run length (available processing time) and memory (population size). A sample of memory use encountered is provided in Figure 7.2(a). For a 48 note piece it is found that approximately 12MB is used by each thousand individuals in a population. While this does not map directly to the size used by each individual (since at least two populations will be alive at once, the current and next populations, dependent on Java's garbage collection facility removing old populations efficiently) it does indicate a size in the order of kilobytes for each individual. Considering the small numbers used within the domain and the relatively large upper limit of the Integer data type in Java this is not a very efficient design.

Having realised this was an issue reference was made to a Java text [CH05] to find out about alternative integer data types. The `int` data type used at this point was a 4 byte data structure storing signed numbers of magnitude near 2 billion. An alternative structure was found in the `byte` data structure, which uses a single byte and stores -128 to 127. For frets, fingers and the like this is more than enough. Extensive optimisations were determined unnecessary, given that saving 3 bytes per individual will save $3 \times 100$k $\approx 300$KB overall in a 100k population, which is trivially small. Even optimising the gene structures making up individuals does not result in significant savings, as demonstrated in Figure 7.2(b).

It appears from this brief optimisation work that this large memory usage is a direct consequence of the decision to use an object oriented data structure approach rather than low level list or array structures. Further work investigating further optimisation of the existing code, or re-implementation of data structures (`Arrangement` and `FingeredNote` classes) in more primitive data types would be beneficial in allowing experimentation with greater population sizes, yielding better performance due to increased genetic diversity. This work should be done in such a way that the interfacing with the rest of the system is unchanged, unless deemed to give additional high-level expressiveness.

Investigate code optimisations to reduce memory use, enabling higher population sizes. Currently memory use is 2 concurrent populations, can this be reduced to nearer to 1 population by redesigning the breeding process. For example remove repetition, instead having an instances in population counter for each individual. This will slow execution (equality test, may be sped up using a hash based population collection) but reduce memory use. It will also require selection recoding to use the new collection.

Can parent pairings be selected and all unselected individuals discarded before creating successor generation, and parents with no future parenting role discarded during the reproduction process. This should trim down the population kept in memory to the few fittest parents.

## 7.13 Nonlinear Cost Functions and Machine Learning

The cost functions produced do not make significant use of available timing information or the information available as a result of eschewing Adjacent Note Assumption and Phrasal Independence Assumption. This shortcoming should be resolved in further work. For those cost functions produced there are few reasons to discount the Adjacent Note Assumption and perform a complete search. Issues may arise with open strings in model 5.3.2 but this problem seems slight.

Taking advantage of this additional information could help solve an issue in training cost functions with divergent types of musical score: the expected behaviour for scale playing is quite different to chord playing or to playing real music where timing is important. With a sufficiently complex cost function able to observe the entire piece at once, parameterised, optimisation of such a cost function through machine learning approaches as attempted by TP and RD seem more likely to succeed as a general solution.

Indeed training cost functions has been the focus of previous works and as such a number of starting points and ideas may be found in the papers considered during the literature review in Chapter 3.

## 7.14 Speech Technology Techniques

Through the project and during related discussions it has become apparent that many of the issues seen in this domain have parallels, if not direct analogues, in the field of automatic

speech recognition. In the past speech research was interested in splitting up speech into constituent segments which could be processed separately, much the same as the segmentation issue with musical phrasing. Again in speech these segments are often not clear: in music and speech there may or may not be pauses which give clear indications, and where there are not other methods must be investigated.

A consideration of applying Hidden Markov Models and similar technologies to this domain, that is generative statistical methods, would be an interesting approach which does not appear in the literature seen during this project. Starting points for such work could be the general speech technology texts [GM00, DJ08].

# Chapter 8

# Conclusion

This project initially set out to investigate the feasibility of a full automatic transcription system, from audio input to playable guitar tablature. After an initial literature review it was decided to focus on the Left-hand fingering problem and further reading was done in this area. A GA approach was determined to be an interesting approach to determining optimal fingering sequences, and indeed was found to have been applied to the problem recently by Daniel Tuohy's Masters Thesis and related publications [TP06a, TP06b, TP04, Tuo06, TPC06].

Through the duration of the project a GA, FINGAR, was implemented in parallel with further reading and a developing understanding of GAs and the problem domain. The behaviour-driven approach taken to unit testing and implementation meant that as the system changed and evolved little was thrown away, aside from an initial misconception about the concept of octaves which resulted in an initial prototype domain encoding being abandoned. This approach was successful in producing a functional implementation verifiably resilient to change. Unit testing was found to be invaluable, though the nuances of testing a random process were not considered early on, and added in to the later tests in an ad-hoc basis.

During the literature survey important simplifications were identified across the projects and coined as the Adjacent Note Assumption and Phrasal Independence Assumption, seeking parallels in related fields of study to understand and explain the issues. These were identified quite late on during the project, which did not leave a great deal of time to plan, develop and execute cost functions and experiments to take advantage of the additional data provided by not using these simplifications. In this respect the project could have gone further, and there is much scope for additional work. From the experiments performed it began to appear that segmenting pieces is desirable in reducing required computation time and hardware requirements, and that the Phrasal Independence Assumption is an issue requiring further investigation. With a more evolved GA in terms of algorithm structure, parallelisation, data encoding efficiency, and distribution and control of processing and results, the computational tractability may improve to an extent where longer pieces of music become tractable. Of course, when the advance to polyphony is made these issues will need to be re-evaluated, and are only likely to worsen given the vastly increased size and complexity of the search space.

Interdisciplinary study seems quite appropriate for the guitar fingering problem. Parallels with early work in speech processing have been observed, and one wonders what insights and approaches might be gleaned from this more developed field. Looking closer to the problem, and further afield from Computer Science, a trained guitarist, particularly an interested music arts or music engineering and technology student, would be an asset in developing the cost functions required by optimisation systems and useful in evaluating results from different systems.

For the author lessons were learned regarding planning ahead for experimentation when

working to a deadline. Results analysis, equipment limitations and tooling issues arising required either unexpected additional work, omission, and generally could have been resolved if discovered earlier.

The project as a whole reinforced concepts of machine learning, graph theory and optimisation, and introduced the previously unseen genetic algorithms. The experience of finding, digesting and disseminating research papers was taxing but very rewarding. On the whole the project is considered a success in so far as providing an introduction to the problem area to any individual who might continue the research with a similar project, or indeed further develop this system given its open source nature.

# Appendix A

# Acronyms

**ASR** Automatic Speech Recognition. 7, 15, 25

**CPF** cumulative probability function. 37, 38

**DAG** Directed acyclic graph. 14, 17

**DP** dynamic programming. 14, 16

**GA** genetic algorithm. 14, 17–19, 22, 23, 25, 27, 28, 30, 34–36, 38, 43, 45, 46, 48, 49, 54, 56, 65

**HM** Heijink and Meulenbroek. 7, 8, 11, 23, 42, 43

**HP** Huang and Panne. 14, 24

**KCT** Kim et al. 24

**LHFD** Left-hand fingering data. 1, 5–7, 11, 23, 25–30, 38

**LHP** left-hand position. 8, 16, 25, 38–43

**LTI** Linear time invariant. 17

**MY** Miura and Yanagida. 10–13, 17, 26

**NN** Neural Network. 23, 24, 31

**PD** probability distribution. 34, 35, 37, 38

**RAL** Radicioni et al. 17, 22, 23, 25, 63

**RD** Radisavljevic and Driessen. 16, 17, 23, 26, 41, 43

**TDD** Test Driven Development. 36

**TP** Tuohy and Potter. 7, 15–17, 22, 23, 25, 28, 43–45, 54

# Appendix B

# Music Terminology

**arpeggio** A sequence of notes from a broken chord played in ascending or descending order [Tay91]. Arpeggiation is closely tied to a right-hand guitar playing technique known as sweep picking, where arpeggios are played at high speed often alternating between ascending and descending runs over a chord.. 16, *see* broken chord

**barre chord** Described in Section 2.5.4. 11, 15, 16, *see* chord

**broken chord** Rather than playing a chord with all notes sounding simultaneously some permutation of the constituent notes is determined and played in sequence [Tay91].. 11, 16, 40, 59, *see* arpeggio

**chord** A group of notes played together. Taylor [Tay91] suggests a definition "Three or more notes sounded together". Many guitarists are less concerned with formal musical definitions and consider two or more notes played together to be a chord, sometimes referred to as 'power chords'. For the purposes of this paper the 2 or more note definition is adopted. The justification for this is that we are not concerned with the number of notes being played, rather in the dichotomy between monophony and polyphony. Where chords are played polyphony is required, unless the chords are arpeggiated or otherwise played as broken chords.. *see* arpeggio & broken chord

**chromatic scale** All notes from a starting note to itself raised or lowered by an octave [Tay91]. *see* octave

**glissando** "a rapid scale passage produced by e.g. drawing a thumb or finger-tip along the white keys of a piano or by sliding the finger along a string of [a] string instrument" [Tay91]. 8, 12, 30

**glissando slide** Described in Section 2.5.2. 16, *see* glissando

**grace note** Optional note(s) in a piece of music (indicated by being smaller) which a player may add or leave out at their preference; they typically increase the difficulty of the piece [Tay91]. 9

**interval** The distance in pitch between two notes, measured in semitones, as part of the chromatic scale. This is equivalent to the number of keys (black and white) moved to the left or right on a piano keyboard to change from one note to the other [Tay91]. 5, 60, *see* chromatic scale & semitone

**left hand fingering data** which fingers are used to play individual notes in the piece. *see* left hand position

**left hand position** The position of the left hand relative to the frets of the guitar neck, typically indicated by the fret which the index finger is resting or playing at. *see* left hand fingering data

**legato** "smooth, indicating no break between notes" [Tay91]. 8

**monophonic** "Music conisting of a single musical line, wthout accompaniment" [McK05]. Music where no more than one note begins at any time in the piece. 11, 26, 27, *see* polyphonic

**monophony** . 17, 24, 28, 30, 40, 52, *see* monophonic

**note** The discrete building block of music theory. A note has an associated frequency (determined by the instrument playing it) and duration (inherent in the note's written presentation). 4, 5, *see* chord

**octave** Octave can take two meanings. As a musical construct it describes the 13 notes (including sharps and flats) from a note to the next note with the same name at a higher or lower pitch. It is also used as a distance metric, being the interval between a note and itself raised or lowered by one octave.. 4–6

**open string** A string played without any left-hand fingers holding it against the fretboard. The '0th' fret.. 40

**polyphonic** "producing many sounds simultaneously; many-voiced. Capable of producing more than one note at a time." [McK05]. 16, 26, *see* monophonic

**polyphony** . 14, 24, 26–28, 30, 34, 52, 56, *see* polyphonic

**score** Written instructions on how to play a piece of music. 1

**semitone** "The distance in pitch (or interval between any note and its nearest neighbour, black or white." [Tay91]. For example the interval in C $\rightarrow$ C# is 1 semitone, and in C $\rightarrow$ D is 2 semitones. 5, *see* interval

**sight-reading** playing for the first time. 12

# Appendix C

# Genetic Biology Terminology

**allele** "one of two or more alternative forms of a gene that arise by mutation and are found at the same place on a chromosome."[McK05]. *see* feature value

**alphabet** The position of a character or substring in a string, where the string represents a chromosome and the substring typically represents a feature (gene). *see* locus

**alternative solution** A structure in its context, for example a produced tablature being played on a guitar. *see* phenotype & structure

**chromosome** "a threadlike structure of nucleic acids and protein found in the nucleus of most living cells, carrying genetic information in the form of genes."[McK05]. An encoding of genetic data. Several combine to form a *genotype*, which interacts with its environment as a phenotype. *see* string

**environment** "the surroundings or conditions in which a person, animal, or plant lives or operates."[McK05]. 61

**feature** A portion of the string (chromosome) which represents a problem solution parameter. 61, *see* gene

**feature value** The value of a particular feature pertinent to the system being modelled, for example the value of a coefficient. *see* allele

**gene** Origin: Biology "a unit of heredity that is transferred from a parent to offspring and is held to determine some characteristic of the offspring"[McK05]. 61, *see* feature

**genotype** "the genetic constitution of an individual organism"[McK05]. *see* structure

**individual** Synonymous with structure. 61

**locus** "the position of a gene or mutation on a chromosome" [McK05]. 20, *see* alphabet

**phenotype** "the set of observable characteristics of an individual resulting from the interaction of its genotype with the environment."[McK05]. In context this would be a solution being performed on a guitar. 61, *see* alternative solution

**population** A collection of individuals grouped together for comparison purposes. A run consists of a sequence of populations, where they are thought of as parent and successor populations. 35

**run** A lineage of populations for a genetic algorithm, from an initial population to the run's terminal population [Mit99]. 18, 35, 61, 62

**string** An ordered list of symbols from the alphabet.[Gol89]. *see* chromosome

**structure** An individual solution in the space of possible encodings. *see* genotype & individual

**successor population** Given a population at some stage of a run the successor population is the population which comes next, being formed of the current population's offspring. Also referred to as the next population. 34, 35, 61

# Appendix D

# Other Terminology

**adjacent note assumption** The assumption that only the current and next state (note + metadata) need be considered to determine a transition cost. See Sayegh in section 3.2.2. 15, 17, 22, 25, 27, 28, 31, 44, 52, 54, 56

**machine learning** Where a function maps a set of inputs to a set of results, an approximation of this function may be learned from examples of the function's outputs for varying inputs, these examples are referred to as training data. See [Mit97]. 13, 15, 16, 26

**overfitting** Training of a function approximation which results in better accuracy for its training data, but worse accuracy for unseen test data. See [Mit97]. 16, *see* machine learning

**phrasal independence assumption** The assumption that musical phrases in a piece exist in isolation, used by RAL as in Section 3.2.2. 17, 22, 25–28, 53, 54, 56

# Appendix E

# Scores

Figure E.1: C Major Scale, 2nd Octave Ascending

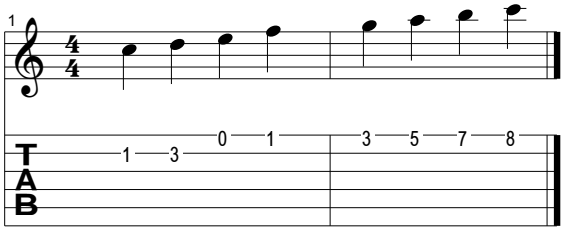
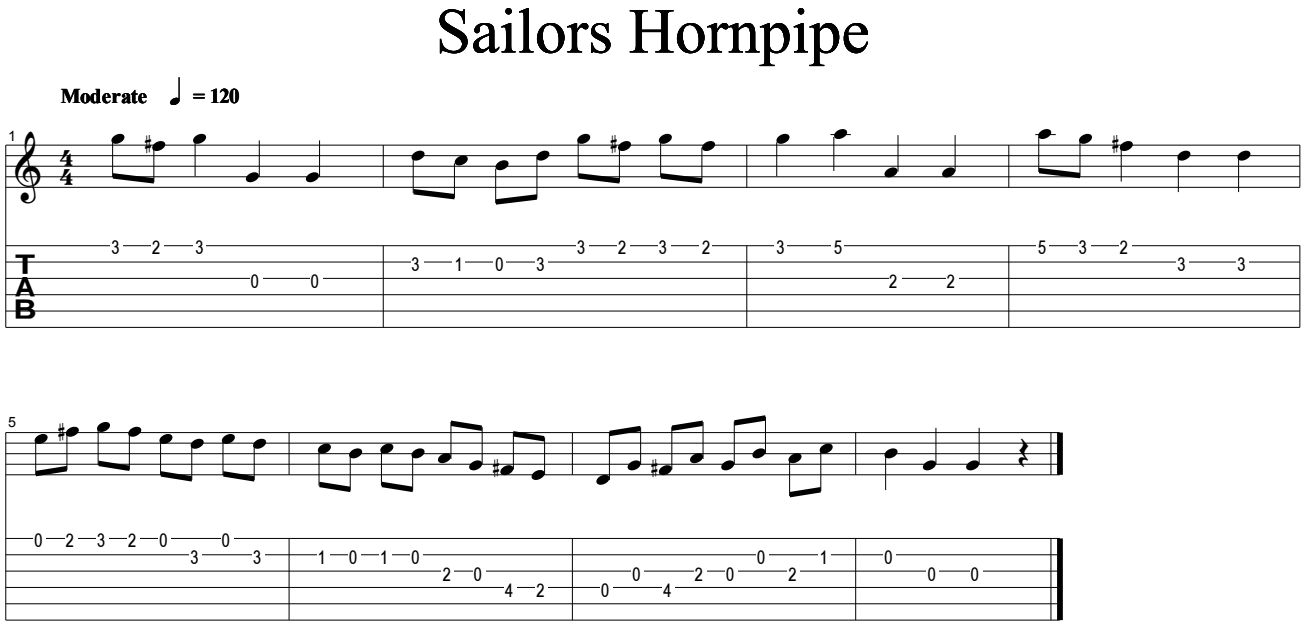


Figure E.2: Sailor's Hornpipe



# Appendix F

# Code Execution

This assumes code has been obtained from the hand-in directory on the departmental network, though alternatively the system may be obtained from `http://www.github.com/nruth/fingar`

## F.1 Binary Execution

To execute the system modify `nruth.fingar.ga.Run.score()` to provide the score you wish to test, some helper methods used during experimentation and testing are available to switch between. In a terminal cd to the FINGAR directory, so that you can see the nruth/ directory containing all Java code. Use the command

```
java -Xmx1g nruth.fingar.Run
```

to execute the system. To see usage instructions for the execution, i.e. GA params, use

```
java nruth.fingar.Run --help
```

An example run from experimentation was begun with:

```
java -Xmx1500m nruth.fingar.Run pop=80000 repeats=11 p_mutate=0.14 sdir=sailors
```

To disable cost value logging per generation change this code in the Run class:

```
FINGAR ga = new FINGAR(score, evolver, farm_sz, true, file.getName(), sdir);
```

to

```
FINGAR ga = new FINGAR(score, evolver, farm_sz, false, file.getName(), sdir);
```

## F.2 Unit Test Execution

JUnit 4 tests were executed with Eclipse, requiring the JMock and JUnit runtimes to be present on the system. These have been included with the source code submission on the departmental network drive. It is suggested that Eclipse is used and the class nruth.fingar.specs.AllFeatureSpecs be executed as a JUnit test case using Eclipse's test runner.

# Bibliography


[Bec04] Kent Beck. *JUnit Pocket Guide*. O'Reilly Media Inc., 2004.

[CH05] G Cornell C Horstmann. *Core Java 2*. Prentice Hall, seventh edition, 2005.

[DJ08] James H. Martin Daniel Jurafsky. *Speech and Language Processing*. Prentice Hall, 2nd edition, 2008.

[DPRL04] L. Anselma D. P. Radicioni and V. Lombardo. A segmentation-based prototype to compute string instruments fingering. *Procs. of the 1st Conference on Interdisciplinary Musicology (CIM04)*, 2004.

[Dre02] Stuart Dreyfus. Richard bellman on the birth of dynamic programming. *Operations Research*, 50(1):48–51, January 2002.

[EMHY06] N Emura, M Miura, N Hama, and M Yanagida. A system giving the optimal chord-form sequence for playing a guitar. *Acoustical Science and Technology*, Jan 2006.

[Ger07] Judith L. Gersting. *Mathematical Structures for Computer Science*. Freeman, 6th edition, 2007.

[GM00] Ben Gold and Nelson Morgan. *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*. Wiley, Jan 2000.

[Gol89] David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Jan 1989.

[HM02] H Heijink and R Meulenbroek. On the complexity of classical guitar playing: Functional adaptations to task constraints. *Journal of motor behavior*, Jan 2002.

[HvdP96] P Huang and M van de Panne. A planning algorithm for dynamic motions. *7th Eurographics Workshop on Animation and Simulation*, Jan 1996.

[KCMT00] J Kim, F Cordier, and N Magnenat-Thalmann. Neural network-based violinist's hand animation. *Computer Graphics International*, Jan 2000.

[McK05] *The New Oxford American Dictionary*. Jan 2005.

[Mit97] T.M Mitchell. *Machine Learning*. Mcgraw-Hill, Mar 1997.

[Mit99] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1999.

[MY02] M Miura and M Yanagida. Finger-position determination and tablature generation for novice guitar players. *Proceedings of the 7th International Conference on Music Perception and Cognition*, 2002.

[Opp97] Willsky A.S. Nawab S. H. Oppenheim, A.V. *Signals Systems*. Prentice-Hall International, 2nd edition, 1997.

[RD04] A Radisavljevic and P Driessen. Path difference learning for guitar fingering problem. *Proceedings of the International Computer Music Conference*, Jan 2004.

[Say89] S Sayegh. Fingering for string instruments with the optimum path paradigm. *Computer Music Journal*, 6(13):76–84, Jan 1989.

[ST88] S Sayegh and M Tenorio. Inverse viterbi algorithm as learning procedure and application to optimization in the string instrument fingering problem. *IEEE International Conference on Neural Networks.*, Jan 1988.

[Tay91] Eric Taylor. *The AB Guide to Music Theory*, volume 1. Associated Board of the Royal Schools of Music, Jan 1991.

[TP04] D Tuohy and W Potter. A genetic algorithm for the automatic generation of playable guitar tablature. *Proceedings of the International Computer Music Conference*, Jan 2004.

[TP06a] D Tuohy and W Potter. Ga-based music arranging for guitar. *CEC 2006. IEEE Congress on Evolutionary Computation.*, Jan 2006.

[TP06b] D Tuohy and W Potter. Guitar tablature creation with neural networks and distributed genetic search. *Proceedings of the 19th International Conference on IEA/AIE'06, Lecture Notes in Artificial Intelligence-4031, Springer-Verlag, Berlin, pp. 244-253*, 2006.

[TPC06] D Tuohy, W Potter, and A Center. An evolved neural network/hc hybrid for tablature creation in ga-based guitar arranging. *Proceeding of the International Computer Music Conference ICMC'06*, Jan 2006.

[TS00] C Traube and J Smith. Estimating the plucking point on a guitar string. *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFx-00)*, Jan 2000.

[Tuo06] D Tuohy. Creating tablature and arranging music for guitar with genetic algorithms and artificial neural networks. *MSAI Thesis available from Don Potter's page: http://www.cs.uga.edu/ potter/CompIntell/index.html*, 2006.

[Whi94] D Whitley. A genetic algorithm tutorial. *Statistics and Computing*, Jan 1994.

[Win02] Darryl Winston. *The Advanced Guitar Case Scale Book*. Jan 2002.

[WS90] Darrell Whitley and Timothy Starkweather. Genitor ii.: a distributed genetic algorithm. *Journal of Experimental Theoretical Artificial Intelligence*, 2(3):189–214, 1990.