Converting Standard Music Notation to Guitar Tablature using Genetic Algorithm

This Report is submitted in partial fulfillment of the requirements for the
BSc Honours Information Systems and Information Technology (DT249) to the
School of Computing,
Faculty of Science, Dublin Institute of Technology.

Author: Artur Jablonski, D10122479

Supervisor: Pat Browne, School of Computing

Date: 01/09/2012

Abstract:

The subject of this work is to implement a multi objective genetic algorithm that will convert a sheet music in a classical notation to a simplified format known as guitar tablature. The solution is implemented in Python programming language and leverages Distributed Evolutionary Algorithms in Python (DEAP) framework. The algorithm was tested and experimented with using four music scores of increasing complexity and the results are presented.

Table of Contents

Figure Index

Example Index

1 Introduction

In this section the aims and objectives of the project are presented together with motivation for choosing this particular subject. A high level overview of proposed software solution is outlined and the structure of this document is explained in details.

1.1 Project objectives

The aim of this project is to create a software system that will generate high quality, playable guitar tablatures from standard notation music using a genetic algorithm to achieve this task. The problem domain will be explained in detail in this document. For now it is sufficient to note that this is an optimization task that will be attempted using biology inspired algorithms collectively called genetic algorithms (GA) that are a part of larger field of Evolutionary Computation (EC). There is a number of objectives that are associated with the project:

1. Implementing genetic algorithm using Distributed Evolutionary Algorithms in Python (Fortin, Rainville, Gardner, Parizeau, & Gagné, 2012) genetic algorithms framework implemented in Python programming language and further referred to in this document as DEAP.

2. Finding an appropriate representation of standard music notation and guitar tablature in the context of genetic algorithms.
3. Finding appropriate fitness function to evaluate quality of a tablature

   Any optimisation method works on the basis of evaluating a fitness function of solution candidates. In case of guitar tablature, a fitness function must be constructed that will take representation of a tablature and produce a numeric value which will correspond to quality of the tablature.
4. Finding appropriate benchmark for generated solutions

The aim of this project is to generate high quality tablatures, but this term is a rather subjective one. There is no agreement as to what constitutes the best tablature from all possible ones. Still, the tablatures generated by the system should be evaluated in some way.

The following items are out of scope for this project

1. Providing production ready user interface or any friendly interface for that matter.
2. Using standardized and interchangeable data structures to represent music notation.
3. Production acceptable performance – it is acceptable for the solution to be slow.

1.2 Personal objectives

There are two important personal objectives behind this project. First and foremost the author wished to gain in depth knowledge about genetic algorithms, their usage, advantages and limitations. A secondary personal objective was to become familiar with Python programming language. The project was perceived as a perfect opportunity to cover both objectives. Firstly, since the solution involves genetic algorithms, it was essential to understand the principles they are based on and secondly, the genetic algorithm framework chosen for this project is implemented in Python programming language and working knowledge of it was required to complete the project.

1.3 Motivation for choosing this topic

The author of this report is an amateur, self taught guitar player. During the journey of learning to play the instrument he found many music scores in classical music notation that he struggled to read. If there was a way of automating translation from such classical music notation to a simplified notation called guitar tablature, the journey would have been less frustrating and more enjoyable. It is hoped, that creating

such software could help other aspiring guitar players (or any fretted stringed instrument players for that matter) without formal music education to speed up the learning process.

1.4 Proposed software solution

The problem of automated translation of classically notated music for guitar to simplified notation will be tackled in this project using an optimisation algorithms under common name – genetic algorithms. These algorithms are inspired by biological process of evolution by natural selection described by Darwin. The algorithms are of course very simplistic comparing to the biological processes. They abstract the essence of evolutionary mechanisms and apply them to solving optimisation problems. The particular genetic algorithms framework used for this task is called DEAP and is implemented in Python programming language.

1.5 Report Overview

This report is organized as follows:

- This section is an overview of the report providing information about project objectives and motivation of choosing this particular topic.
- Section 2 “Problem domain” describes in detail what problem this project is trying to tackle. It is essential to understand its content in order to understand the following parts of this report.
- Section 3 “Genetic algorithms” explains the 'anatomy' of genetic algorithms in details.
- Section 4 “Development environment” details the development environment setup used including an overview of Python programming language and DEAP genetic algorithm framework.
- Section 5 “Solution domain” details the approach of finding the solution including tuning genetic algorithm parameters, fitness function implementation as well as evaluation of results.

- Section 6 “Conclusion” summarizes what was achieved in the project and what are potential further steps that could improve the solution.
- Section 7 “Appendices” contains input scores used during testing phase of the project, generated tablatures as well as full source code that was developed as a part of this project.

2 Problem domain

Anybody trying to learn how to play guitar, mandolin, banjo or any other fretted string instrument will quickly realize an inherent quality of the instrument which adds to complexity of the learning process – most of notes that can be played on these instruments can be produced in more than one position (i.e. location on the fretboard of the instrument). For simplicity reasons let us consider a standard 6 string, 20 frets guitar in standard tuning. A range of notes that can be produced from such guitar comparing to other instruments is shown in the Fig 1. The top section is presenting the range in a standard music notation. Below is the same range presented in guitar tablature notation. Below that there is a representation of piano keyboard for reference and comparison to other instruments ranges.

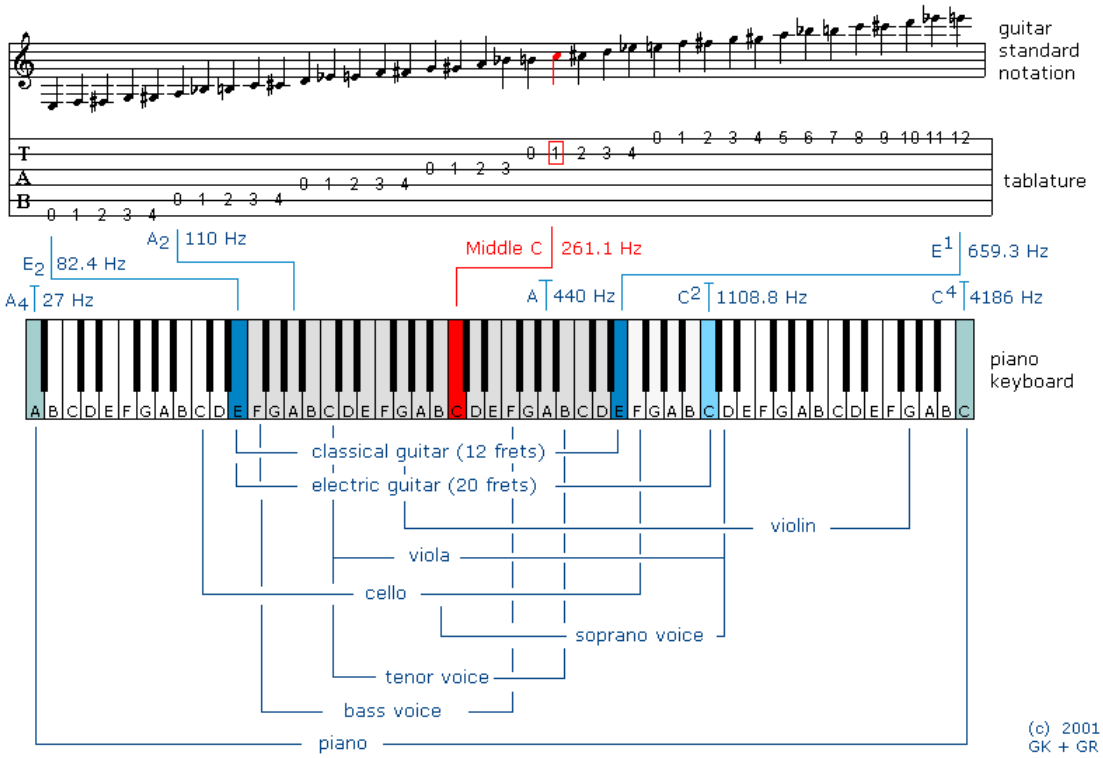


Fig 1: Range of 20 fret electric guitar (from http://12bar.de/basics.php)

In standard tuning of this instrument a middle 'C' (shown in red on the piano keyboard in Fig 1) can be played in 5 different positions (i.e. locations on the fretboard). First 3 positions are shown with black circles in Fig 2. Remaining two positions are on 15th fret of the 5th string and 20th fret of the 6th string and are not visible in the Fig 2 since it only covers first fourteen frets of the instrument.

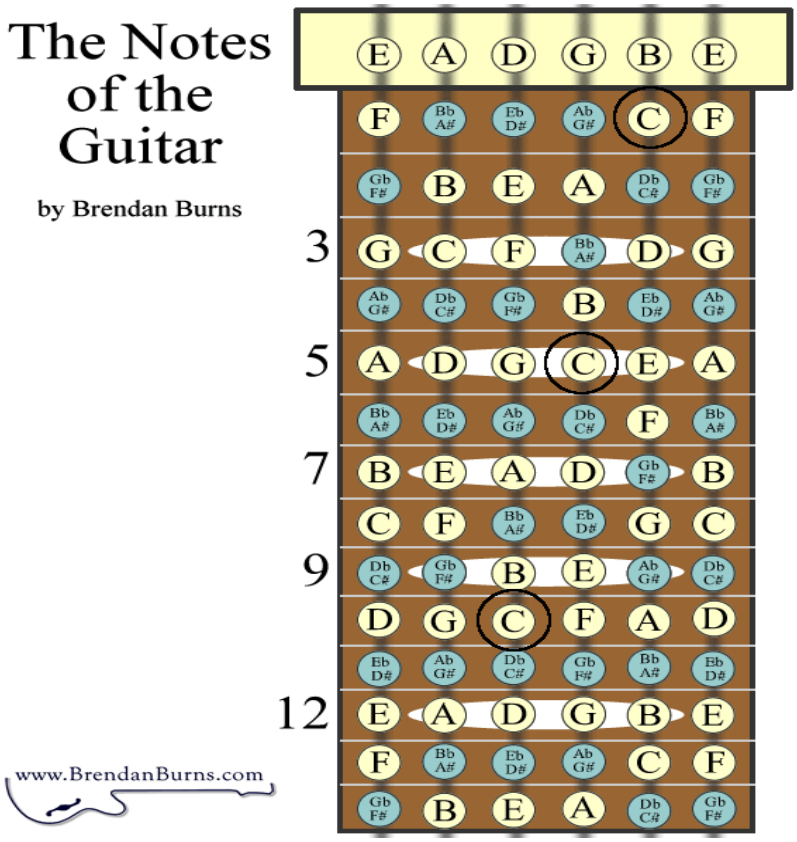


Fig 2: Notes covering first 14 frets of guitar (from http://www.brendanburns.com)

All the Cs in black circles are exactly the same pitch (the strings depressed at the given locations will vibrate with the same frequency). This quality is central to the problem that this project is trying to tackle. Unless a music score in a standard notation is annotated with some additional information about position (i.e. location on the fretboard) that each note needs to be played at, there exists a number of possible ways this score can be played. This number grows exponentially with number of notes that a particular music score consists of. The mentioned 20 fret guitar covers a range of almost 4 octaves. Each octave consists of 12 different pitches which adds up to 48 distinct pitches that such instrument can produce. There are however 126 different positions that can be used (20 frets plus an open position on each of the 6 strings) which means that on an average there are approximately 3 different positions that each note can be played at. Let us consider a rather short piece of music consisting of 12 bars each having 6 notes. A simple calculation will bring us to an intimidating

number of approximately $20*10^{33}$ possible fingerings these 72 consecutive notes can be played in. Most of the possible “paths” a guitar player's fingers can follow to play the score will not make much pragmatic sense because of human hand physiology constraints (i.e. consecutive notes can be set too further apart or be too awkwardly placed to be played by a person). Still, the best that a guitar learner can do is trial and error attempts to find a playable fingering for a music score that doesn't explicitly indicate a position to use.

Because many starting guitar players without formal education in music find the standard music notation difficult to learn (not to mention the problem explained in the previous paragraph) a simplified music notation for guitar was invented. It is referred to as guitar tablature and it is a predominant format of guitar music notation available through various internet services. Instead of staff and notes the notation uses a representation of 6 guitar strings and each note is represented as a number on one of them. The number indicated is a fret over which the string should be pressed in order to produce a particular pitch. A simple example of tablature is shown in Fig 3

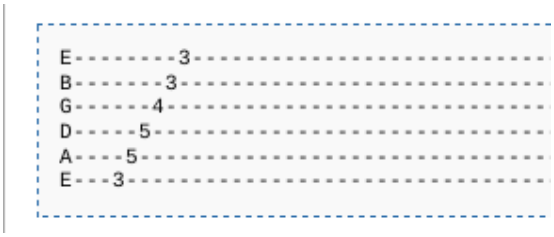


Fig 3: An example of guitar tablature (author)

Note that tablature does not indicate with which finger to press the string. There are additional elements of the notation that detail the way the note should be played. An excellent documentation on the tablature notation can be found on the Web at http://www.tabwiki.com/index.php/Main_Page (“TabWiki,” n.d.). Both the standard and tablature notations are time based, i.e. you can imagine a time line running from left to right. Notes are played in that order and if they happen to occur stacked vertically, they are played simultaneously. That is where similarities between the two notations end. The tablature notation is inferior in many ways. There is no concept of

a bar (measure), the duration of each note is not specified, there is no time signature or key signature, to name a few. The question that this simple notation answers is: which string and on which fret to press to produce sounds. For this precise reason there is no ambiguity in it as to position choice.

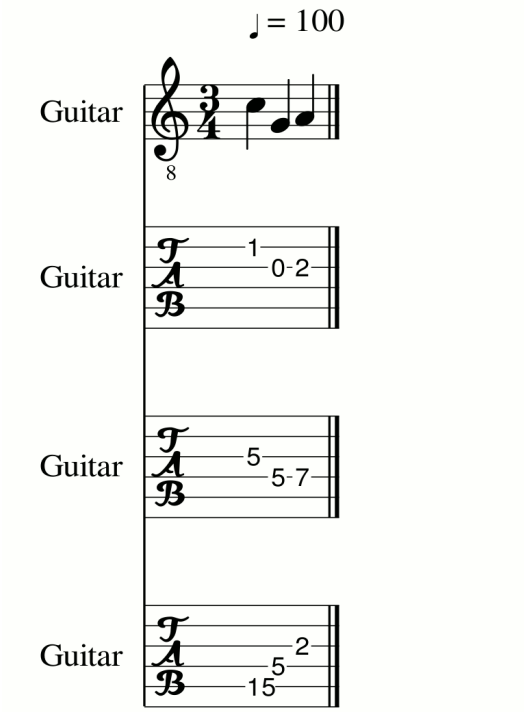


Fig 4: Standard notation and 3 equivalent tablatures (author)

Based on the information presented so far, it can be said that for each standard notation music score consisting of *n* notes there are approximately 3^n equivalent tablature notations. The Fig 4 shows 3 of approximately 27 equivalent tablatures corresponding to a measure consisting of 3 notes only.

This project focuses on computer aided generation of high quality tablatures from standard music notation. There are two problems associated with finding a high quality tablature among this exponentially growing space of possible solutions. Firstly, 'high quality' of a tablature is a rather subjective term. Obviously most of guitar players would probably agree on 'low quality' tablatures, but given a set of

'good quality' tablatures of the same piece of music, there would certainly be diversity of opinions as to which one is the best. It is to a certain extent a matter of preferrence. So the first challenge is to define criteria of a high quality tablature. The second problem is how to search for the best tablature given a set of criteria. Extensive search quickly becomes not practical as number of notes in a music score grow. The approach this project will take is to use a genetic algorithm to find an optimal solution to the problem. Genetic algorithms are described in the next section.

3 Genetic algorithms

Genetic algorithms are part of Evolutionary Computation (EC) which is a common name for search and optimisation algorithms that are inspired by Darwinian evolution of living organisms and are a perfect example of cross-disciplinary approach to computing. The most popular technique in the Evolutionary Computation research has been the genetic algorithm (Sivanandam & Deepa, 2010). There are various implementation approaches to a genetic algorithm. Typically, such algorithm goes through the following stages: initializing randomly a population of candidate solutions, selecting candidates for 'reproduction', creating next generation of solutions by applying genetic operators (for example: crossover, mutation) (Mitchell, 1998). The steps are then repeated for the new generation. The algorithm stop condition may vary depending on particular implementation. A detailed flow of genetic algorithm is shown in Fig 5.

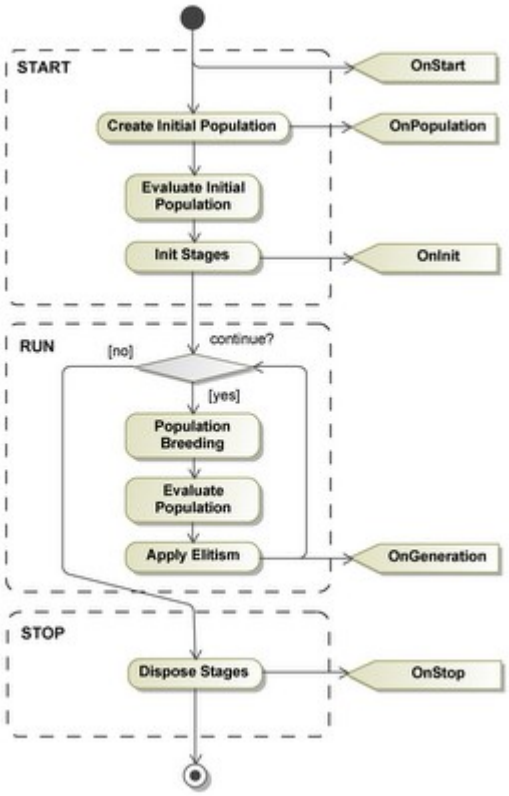


Fig 5: Stages of genetic algorithm (from http://jenes.ciselab.org/tutorials/anotomy)

Genetic algorithms offer an alternative method of optimisation that display performance improvements over enumerative, calculus based and random searches (Goldberg, 1989) and therefore have been chosen for this project.

It is important to understand that a term genetic algorithm designates a class of algorithms and not any particular implementation of it. Therefore the algorithms differ between themselves in the way they handle the different stages of the generic genetic algorithm described in the previous paragraph. The following sections look closer at different stages of a generalized genetic algorithm and how they can be approached in a concrete manifestation of it.

3.1 Encoding

Encoding is the very first problem to solve when trying to use a genetic algorithm for an optimisation problem. In essence, encoding is how an instance of solution is represented. In genetic algorithms the terms gene and chromosome are used interchangeably and they designate a particular representation of a solution to the problem that the genetic algorithm is trying to solve. For example, assuming that the problem that we're trying to solve is the Travelling Salesman Problem (TSP)*, where each city has a two dimensional Cartesian coordinates, then an example of a gene that would represent one solution of this problem would be a sequence of coordinates of subsequently visited cities where each city can only occur once and all the cities are listed in the solution. Let us assume that there are 3 cities in the TSP problem: A(0,0), B(7,7), and C(23,14), then a solution could be represented as:

Solution A: (0,0),(23,14),(7,7)

The representation needs to be adequate in terms of the domain space, but it also needs to be designed in a way that allows the genetic operators to be applied. Some forms of encoding are explained in the following sections.

3.1.1 Binary Encoding

Binary encoding is the most common, mainly because first works about genetic algorithms used this type of encoding. In binary encoding every chromosome is a string of bits, 0 or 1.

Chromosome A	10110010110010101110010 1
Chromosome B	11111110000011000001111 1

Fig 6: Example of chromosomes with binary encoding (from http://www.obitko.com/tutorials/genetic-algorithms/encoding.php)

Binary encoding gives many possible chromosomes even with short length of the binary string. On the other hand, this encoding is often not natural for many problems

* Travelling Salesman Problem is a well known and thoroughly studied problem in optimisation. It can be defined as a problem of finding an optimal (shortest) path between a number of cities so that each city is only visited once and all cities are visited.

and sometimes corrections must be made after crossover and/or mutation. This representation is suitable for Knapsack problem, where there are number of objects of given value and size as well as a knapsack with a given capacity. The optimisation problem is to find such combination of objects that fit the knapsack and have maximum value. Each bit in a chromosome designates an object. 0 means that the object is not in the knapsack and 1 that it is.

3.1.2 Permutation Encoding

Permutation encoding can be used in ordering problems, such as travelling salesman problem or task ordering problem. In permutation encoding, every chromosome is a string of numbers, which represents number in a sequence.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Fig 7: Example of chromosomes with permutation encoding (from http://www.obitko.com/tutorials/genetic-algorithms/encoding.php)

Permutation encoding is only useful for ordering problems. Even for this problems for some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e. have a valid sequence in it).

3.1.3 Value Encoding

Direct value encoding can be used in problems, where some complicated value, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult. In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Fig 8: Example of chromosomes with value encoding (from

http://www.obitko.com/tutorials/genetic-algorithms/encoding.php)

Value encoding is very good for some special problems. On the other hand, for this encoding is often necessary to develop some new crossover and mutation specific for the problem. The representation of tablature in this project is using value encoding.

3.1.4 Tree Encoding

Tree encoding is used mainly for evolving programs or expressions for genetic programming (another branch of Evolutionary Computing). In tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language.

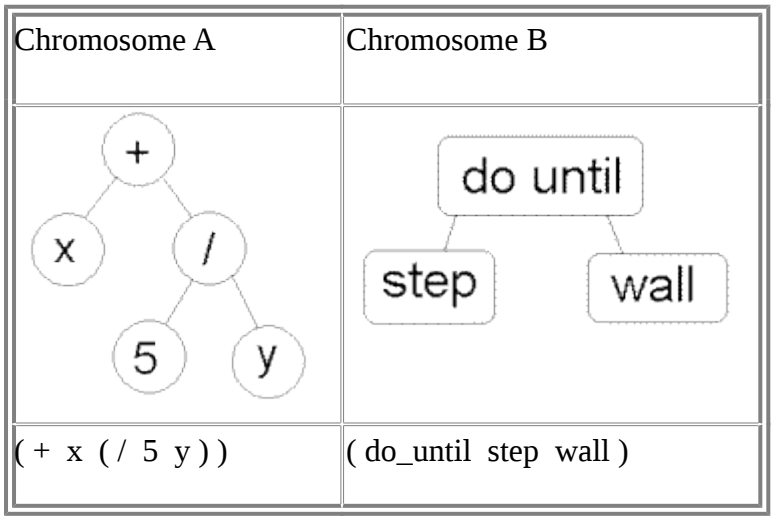


Fig 9: Example of chromosomes with tree encoding (http://www.obitko.com/tutorials/genetic-algorithms/encoding.php)

Tree encoding is good for evolving programs. Programming language LISP is often used to this, because programs in it are represented in this form and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

3.2 Initialization

The first step in a genetic algorithm is to initialize the solution population. This usually involves some randomness, but it is important to understand that each

individual in the population needs to be a valid solution. In the case of the tablatures, that means that each tablature in the initial population must be equivalent to the piece of music notated in a classical way that the tablature is to represent. The size of the population can vary.

3.3 Selection

Selection is a process of choosing certain individuals for the reproduction stage of the algorithm. The process is based in one way or another on a fitness of each of the individuals which is calculated based on the fitness function. Fitness function is a function that evaluates a solution according to some criteria. Applying the function to all individuals in a population of solutions allows for ordering them from 'the best' individuals to 'the worse' ones. In genetic algorithms, the best individuals are chosen to reproduce and create new populations of solutions that have better chance to score higher in the fitness function. In the example of TSP problem, the function is simply a distance that is covered by travelling through a particular sequence of cities. In the TSP problem, the individuals with lower distance are better solutions since the problem is to minimize the distance travelled. The fitness value is often normalized so that the sum of fitness values for all individuals equals 1. The individuals are then sorted by descending fitness values and accumulated normalized fitness values are calculated for each individual. The accumulated normalized value is a sum of an individual's fitness value and all other individuals that precede it in the fitness value descending ordering. Next, a random number is generated within a [0;1] range and the individual chosen is the first one with its accumulated value larger than the generated number. If this procedure is repeated until there are enough selected individuals, this selection method is called fitness proportionate selection or roulette-wheel selection. If instead of a single pointer spun multiple times, there are multiple, equally spaced pointers on a wheel that is spun once, it is called stochastic universal sampling. Repeatedly selecting the best individual of a randomly chosen subset is tournament selection. Taking the best half, third or another proportion of the individuals is truncation selection.

There are other selection algorithms that do not consider all individuals for selection,

but only those with a fitness value that is higher than a given (arbitrary) constant. Other algorithms select from a restricted pool where only a certain percentage of the individuals are allowed, based on fitness value.

Retaining the best individuals in a generation unchanged in the next generation, is called elitism or elitist selection. It is a successful (slight) variant of the general process of constructing new population.

3.4 Applying genetic operators

The next step of the algorithm is to create a second generation of solutions from the ones chosen for reproduction in the step of selection described in the previous section. This process involves applying genetic operators of crossover and mutation. A child solution is created based on two parent solutions. The important thing to understand is that the new solution will share many of the characteristics of its parents. Also, each of the generated new individuals must be a valid solution to the given problem. This means that the operators must be designed so that they do not invalidate solutions. Some genetic algorithms use more than two parent solutions to generate an offspring. The process of applying the operators leads ultimately to generating a new population that is different from the previous one. In general an average fitness value of the new population will be higher than that of the previous one. This is because mostly the best (in terms of the fitness value) individuals are chosen for breeding. However some less fit individuals are chosen as well, this is to prevent the algorithm from converging on a local optimum. The following sections describe crossover and mutation operators in more detail:

3.4.1 Crossover

Crossover is a genetic operator that is used to produce new solutions based on existing solutions by exchanging fragments of chromosomes. The assumption is that combining fragments of good solutions may lead to creating even better solutions. The simplest example of crossover is a one point crossover. The operator is used in the reproduction phase of genetic algorithm to create new population of solution based on selected solutions of current population. Depending on representation of

particular solution a crossover can result in invalid solutions and extra caution needs to be taken to implement crossover operator properly. For example in TSP problem exchanging parts of individual solutions between valid solutions can result in an invalid solutions to the problem (solutions with duplicate cities and/or missing cities in the travelling path). Assuming that a solution is represented as a string of characters, then a simple form of crossover can be illustrated by the following example where symbol '|' is used to show the crossover point:

```
input:
solution A: art|ur
solution B: dav|id

crossover output:
solution A': artid
solution B': davur
```

In essence, crossover operator is used to combine parts of parent solutions to create a child solution. There are different techniques for achieving this goal. Some of them are detailed in the following sections.

One-point crossover technique is based on selecting a single point on both parents' chromosomes. All data beyond that point in either organism chromosome is swapped between the two parent organisms. The results are the children:

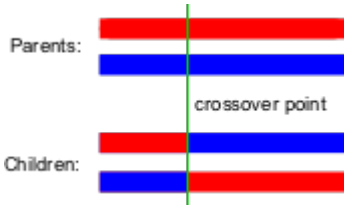


Fig 10: One-point crossover (from http://wikipedia.org)

Two-point crossover technique is based on selecting two points on both parents' chromosomes. All data between the two points is then swapped between the two parent organisms.

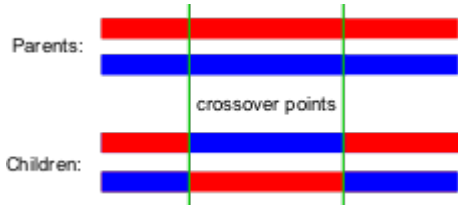


Fig 11: Two-point crossover (from http://wikipedia.org)

Cut and splice crossover is another variant of the crossover operator. It may not be suitable for all problems as it results in an offspring with differing length of chromosomes. In essence this variant is similar to one-point crossover, but with the crossover point chosen independently on both parents

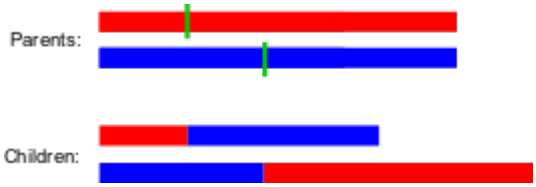


Fig 12: Cut and splice crossover (from http://wikipedia.org)

Uniform Crossover and Half Uniform Crossover variant of crossover operator allows parents to contribute a level of genes as opposed to a fixed segment of genes. A mixing ratio is chosen that designates a level of gene contribution, for example a mixing ratio of 0.5 means that the offspring has approximately half of the genes from first parent and the other half from second parent.

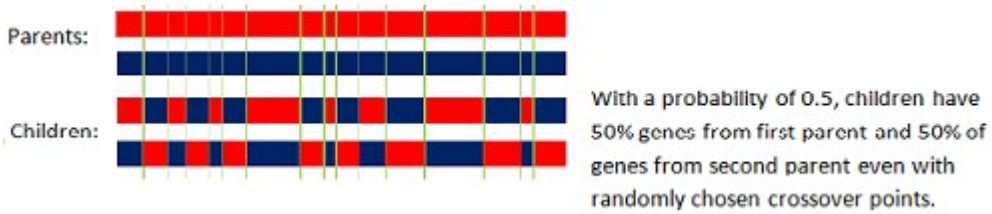


Fig 13: Uniform crossover with mixing ratio 0.5 (from http://wikipedia.org)

3.4.2 Mutation

Mutation is another genetic operator that is used to introduce diversity in the population of solutions. It is usually carried out after crossover operator with a particular probability rate. One of side effects of applying the mutation operator is that the population will explore the search space and reduce a risk of diverging on local minimum/maximum point. As a result of stochastic application of the mutation, it is not guaranteed to be carried out on every solution. The operator is dependent on the characteristics of the gene representation and constraints of the problem. Similarly to the crossover operator, an additional steps may be required in order not to produce invalid solutions. In its simplest form the mutation can be changing single element of chromosome to some random value. Assuming that a solution is represented as sting of characters, then a mutation can be illustrated by the following example where the position that the mutation occurs is shown in bold.

Input:
solution A: art**u**r

mutation output:
solution A': art**e**r

In case of TSP problem a simple change of one city to another would produce an invalid solution since two cities would be listed in a solution and one would be missing. Therefore in this particular case mutation could take a form of swapping two randomly chosen cities in particular solution.

Mutation operator is used to maintain genetic diversity from one generation of a population of solutions to the next. It is analogous to biological mutation in that it may result in a chromosome that manifests itself in a better adapted organism, or translating this statement to the realm of optimisation algorithms, to more optimal solution. Mutation is an important part of the genetic search as it helps to prevent the population from stagnating at any local optima. Mutation occurs during evolution according to a user-definable mutation probability. This probability should usually be set fairly low (0.01 is a good first choice). If it is set to high, the search will turn into a primitive random search, if it is set too low the population of solutions will stagnate

and will not explore the search space. There are several forms that mutation can take. These are explained in the next paragraph.

Flip Bit -A mutation operator that simply inverts the value of the chosen gene (0 goes to 1 and 1 goes to 0). This mutation operator can only be used for binary genes.

Boundary - A mutation operator that replaces the value of the chosen gene with either the upper or lower bound for that gene (chosen randomly). This mutation operator can only be used for integer and float genes.

Non-Uniform - A mutation operator that increases the probability that the amount of the mutation will be close to 0 as the generation number increases. This mutation operator keeps the population from stagnating in the early stages of the evolution then allows the genetic algorithm to fine tune the solution in the later stages of evolution. This mutation operator can only be used for integer and float genes.

Uniform - A mutation operator that replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.

Gaussian - A mutation operator that adds a unit Gaussian distributed random value to the chosen gene. The new gene value is clipped if it falls outside of the user-specified lower or upper bounds for that gene. This mutation operator can only be used for integer and float genes.

3.5 Termination

The process of selecting parents and breeding new solutions is repeated until a termination condition has been reached. There are various ways the termination condition can be expressed, some common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached

- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

3.5.1 Multi objective optimisation

In a basic form of optimisation using genetic algorithms, individual solutions are evaluated using a single objective fitness function. An example of this could be the Travelling Salesman Problem mentioned before. In a more complicated scenario, where individual's fitness entails many objectives, the different parts of genetic algorithm need to reflect this. In particular, selection part of the algorithm needs to select individuals for 'breeding' taking into consideration multiple objectives of the fitness function. The multi objective selection algorithm used in this project is called SPEA2 and is implemented by the DEAP framework.

4 Development environment

This section explains the development environment setup. Firstly, Python programming language main features are described. Then DEAP (Distributed Evolutionary Algorithms in Python) framework used in this project is explained. The last subsection describes Eclipse IDE with PyDEV plugin.

4.1 Python programming language

Python is object-oriented, interpreted and interactive high-level programming language. It runs on Windows, Linux/Unix, Mac Os X, and has been ported to the Java and .NET virtual machines (“Python Programming Language – Official Website,” n.d.). Python is free to use for both commercial and not-commercial products. According to TIOBE index, which is an indicator of popularity of programming languages, Python is on 8th position as of July 2012.

The language supports a number programming paradigms and doesn't enforce any particular one on developers. There is full support for object-oriented and structured

programming and some support for functional and aspect-oriented programming.

The following sections detail some of the features and idioms in Python programming language. These contain features specific to Python that may be found surprising by developers coming from a strongly typed languages background like Java or C. The list is not meant to be exhaustive, for this there are countless resources available online including the official documentation for Python (the web site can be found in the References section of this report).

4.1.1 Duck typing

Source code written in Python does not declare the types of variables or parameters or methods. This makes the code short and flexible, though there is very little checking at compile time, deferring almost all type, name, etc. checks on each line until that line runs. Python tracks the types of all values at runtime and flags code that does not make sense as it runs. The dynamic typing applies to both primitives and objects and the particular type of dynamic typing in Python is referred to as duck typing, the term which comes from the following phrase:

“When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.”

To translate the phrase into programming language terms, as long as an object has the expected behaviour it can be used regardless of its declared type. Let us take this code snippet as an example.

```
1  class Cat:
2      def giveVoice(self):
3          print "meow"
4
5  class Dog:
6      def giveVoice(self):
7          print "woof"
8
9  def foo(obj):
10     obj.giveVoice()
11
12 aCat = Cat()
13 aDog = Dog()
14
15 foo(aCat)
16 foo(aDog)
```

Example 1: Duck typing in Python

Here two classes are declared *Cat* and *Dog* (lines 1 and 5) that both have *giveVoice* method declared (lines 2 and 6). Then a function *foo* is declared that executes *giveVoice* method on an argument that was passed to it. Please note that the *foo* method is agnostic as to exact type of the argument passed. Then the *foo* method is called first with an instance of *Cat* and then with an instance of *Dog*. This code runs fine even though *Cat* and *Dog* classes have no inheritance relationship between each other.

4.1.2 Indentation based syntax

One unusual Python feature is that the white space indentation of a piece of code affects its meaning. A logical block of statements such as the ones that make up a function should all have the same indentation, set in from the indentation of their parent function, "if" statement, etc. If one of the lines in a group has a different indentation, it is flagged as a syntax error. Other languages usually use curly brackets or keywords (Begin, End) for this purpose. An example of Python's indentation syntax is shown in Example 2. Here the statements following the 'if' are executed if variable *x* evaluates to *True*.

```
if x:
    x += 4
    print x
```

Example 2: Indentation syntax in Python

4.1.3 Data structures

Python comes with a number of build in data types and data structures. Above the typical set of types like boolean, integer and float numbers Python comes with a number of sequence types. These include, but are not limited to: strings, lists and tuples. Tuples and lists have very similar behaviour, but tuples are immutable (their elements cannot be changed once assigned). Sequences can be accessed using indexing or provided API. An interesting variation of index syntax that Python provides is slicing. Some example of slicing syntax together with common idioms related to sequences are provided in Example 3 below with inline comments explaining their meaning.

```
aList = [4,3] #lists are initialized using square brackets
aTuple = (2,4) #tuples use curly brackets

unpack1, unpack2 = aTuple #unpacking of a tuple into two
variables

aLongerList = [1,2,3,4,5,6]
slice = aLongerList[1:5] #extract slice from index 1 through 5
(exclusive)
slice = aLongerList[:3] #extract slice from the beginning
through 3 (exclusive)
slice = aLongerList[3:] #extract slice from index 3 until the
end of the list
copyOfALongerList = aLongerList[:] #extract slice from
beginning until end - a copy

#list comprehension example
#returns a list of squared numbers from 0 to 9
print [i**2 for i in range(10)]

#loopint over two sequences at the same time using zip()
function
#note that both functions are of different types
for a,b in zip(aList, aTuple):
    print a,b
```

Example 3: Slicing syntax in Python

4.2 DEAP

DEAP - Distributed Evolutionary Algorithms in Python (Fortin et al., 2012) is a genetic programming framework implemented in Python programming language that comes with many building blocks of genetic algorithms implemented and allows for easy distribution of the algorithm tasks in a multi-core/multiprocessor environment. It was chosen for this project mainly because of little effort required to parallelize the algorithm. It seeks to make algorithms explicit and data structures transparent. It also incorporates easy parallelism where users need not concern themselves with implementation details like synchronization and load balancing, only functional decomposition. DEAP has a very exhaustive and comprehensive documentation backed by many practical examples that speeds up learning process. The framework consists of two components DTM and EAP.

4.2.1 DTM component of DEAP framework

The DTM component (distributed task manager) of DEAP framework is responsible for spreading workload of the genetic algorithm over several computation units using TCP protocols or MPI (Message Passing Interface) connection. In other words, this part is able to parallelize execution of the implemented optimisation algorithm with minimal code changes required. The DTM supports easy to use parallelization paradigms, offers a multiprocessing interface, covers basic load balancing algorithm and supports TCP communication manager.

This part of the framework was not explicitly used in this project, however it may be used for future development.

4.2.2 EAP component of DEAP framework

The EAP is the evolutionary core of DEAP framework. It provides data structures, methods and tools to design any kind of evolutionary algorithm. The following list contains main features of the framework:

- Genetic algorithm using any imaginable representation
    - List, Array, Set, Dictionary, Tree, Numpy Array, etc.
- Genetic programming using prefix trees
    - Loosely typed, Strongly typed
    - Automatically defined functions
- Evolution strategies (including CMA-ES)
- Multi-objective optimisation (NSGA-II, SPEA-II)
- Co-evolution (cooperative and competitive) of multiple populations
- Parallelization of the evaluations
- Hall of Fame of the best individuals that lived in the population
- Checkpoints that take snapshots of a system regularly
- Benchmarks module containing most common test functions
- Genealogy of an evolution
- Examples of alternative algorithms : Particle Swarm Optimization, Differential Evolution, Estimation of Distribution Algorithm

4.2.3 An example of genetic algorithm in DEAP framework

The following section contains more detailed explanation of how to implement Genetic Algorithm using DEAP framework. This is illustrated by some code snippets that are referred to in text using line numbers.

The core of the architecture of the framework is based on the *creator* module and *Toolbox* class. An example of a simple genetic algorithm implemented in the framework follows together with explanation of different parts of the code. In the example each individual consists of a list of 10 integers. The algorithm searches for an individual with the lowest sum of the integers that make up its genome.

4.2.3.1 Type creation

```
1 from deap import base, creator, tools
2 import random
3
4 creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
5 creator.create("Individual", list, fitness=creator.FitnessMin)
```

Example 4: Simple genetic algorithm in DEAP, part 1

As a first step, the appropriate types need to be created. This is done dynamically and provided by the *creator* module. Line 4 creates *creator.FitnessMin* class that inherits from *base.Fitness* class. The *weights* argument is used to apply weights on fitness criteria. In this case -1.0 means a single criterion fitness will be used and the algorithm will be trying to find a solution with minimal value of the fitness function. On line 5 a *creator.Individual* class is created that will be used as solution representation. This type derives from *list* type and the fitness used for the individual is the one that was created on line 4.

4.2.3.2 Initialization

Now that the types are defined, they need to be initialized with either guessed or random valued. A *base.Toolbox* class provides an easy way of initializing components

```
6 IND_SIZE = 10
7 toolbox = base.Toolbox()
8 toolbox.register("attribute", random.random)
9 toolbox.register("individual", tools.initRepeat,
creator.Individual,
10 toolbox.attribute, n=IND_SIZE)
11 toolbox.register("population", tools.initRepeat, list,
toolbox.individual)
```

Example 5: Simple genetic algorithm in DEAP, part 2

of an algorithm. An instance of the class is created on line 7. The object of this class is then used to register different functions. This is done by using *register* method which takes alias of the function as the first argument, the function itself as a second argument, followed by any arguments that the aliased function requires. Let us imagine there is a function that returns the bigger of two integers:

```
def max(a,b):
      return a if a > b else b
```

Example 6: Simple max function

the *toolbox* instance could then be used to create an alias to the function like this:

```
toolbox(“maxAlias”, max, a, b)
```

Example 7: Function aliasing in DEAP

This alias could then be referenced and called like this:

```
toolbox.maxAlias
toolbox.maxAlias(3,4)
```

Example 8: Referring to aliased function in DEAP

On line 8, an alias to Python’s *random* method is created with name attribute. On line 9 an *individual* alias is created that uses some of the DEAP code to specify initialization of an individual. The semantics of this initialization is that the individual is consisting of 10 random digits. Finally, on line 10 *population* alias is created that will initialize population of individuals described before. The exact size of the population will be decided at the time of initialization of the algorithm.

4.2.3.3 Operators

```
12 def evaluate(individual):
13     return sum(individual),
14
15 toolbox.register("mate", tools.cxTwoPoints)
16 toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1)
17 toolbox.register("select", tools.selTournament, tournsize=3)
18 toolbox.register("evaluate", evaluate)
```

Example 9: Simple genetic algorithm in DEAP, part 3

The operators are just like initializers, except that some are already implemented in the *tools* module. A careful choice of this operators needs to be made that is fit for

particular problem. On line 12 a simple evaluation function is defined. It takes an individual as an argument and returns a sum of the digits it consists of. On lines from 15-18 the operators themselves are registered using the *toolbox* instance. On line 15 a *mate* alias is created that uses a ready implementation of two point crossover. On line 16 a *mutate* alias is created using a ready implementation of Gaussian mutation operator. On line 17 a *select* alias is created that sets a ready implementation of selection tournament strategy for selecting individuals for 'mating'. Finally, line 18 defines an alias to the *evaluate* function defined before with the same name.

4.2.3.4 Algorithm

```
19 def main():
20     pop = toolbox.population(n=50)
21     CXPB, MUTPB, NGEN = 0.5, 0.2, 40
22     # Evaluate the entire population
23     fitnesses = map(toolbox.evaluate, pop)
24     for ind, fit in zip(pop, fitnesses):
25         ind.fitness.values = fit
26 
27     for g in range(NGEN):
28         # Select the next generation individuals
29         offspring = toolbox.select(pop, len(pop))
30         # Clone the selected individuals
31         offspring = map(toolbox.clone, offspring)
32 
33         # Apply crossover and mutation on the offsprings
34         for child1, child2 in zip(offspring[::2],
offspring[1::2]):
35             if random.random() < CXPB:
36                 toolbox.mate(child1, child2)
37                 del child1.fitness.values
38                 del child2.fitness.values
39 
40         for mutant in offsprings:
41             if random.random() < MUTPB:
42                 toolbox.mutate(mutant)
43                 del mutant.fitness.values
44 
45         # Evaluate the individuals with an invalid fitness
46         invalid_ind = [ind for ind in offsprings if not
ind.fitness.valid]
47         fitnesses = map(toolbox.evaluate, invalid_ind)
48         for ind, fit in zip(invalid_ind, fitnesses):
49             ind.fitness.values = fit
50 
51         # The population is entirely replaced by the
offsprings
52         pop[:] = offspring
53 
54     return pop
```

Example 10: Simple genetic algorithm in DEAP, part 4

The rest of the code details the genetic algorithm itself. On line 20 a population of 50 individuals is created. Line 21 defines probabilities of crossover and mutation operators as well as number of generations of the algorithm. On line 23 all individuals are evaluated according to the fitness function. Each individual is assigned its fitness value in the for loop on line 24. On line 27 a main loop of the algorithm starts. It executes 40 times as defined previously in the NGEN variable. One iteration contains

the following logic. On line 29 a tournament selection is performed to choose the best individuals for 'mating'. This individuals are then cloned on line 31 so that the original individuals are not affected. On line 34 the selected individuals are processed in pairs in for loop. Line 35 determines whether crossover operator should be applied. If that is the case, then previously registered two point crossover function is applied to a pair of individuals on line 36. Lines 37 and 38 removes fitness values from the individuals as they are no longer accurate. On line 40 the selected and potentially crossed over individuals are processed one at a time. Line 51 determines whether mutation operator should be applied. If this is the case, then the previously registered Gaussian mutation operator is applied to the individual and on line 52 its fitness value is removed as it is no longer valid. Lines 46-50 re-evaluate these individuals whose fitness is no longer valid. Line 52 replaces the population with the new generation and the algorithm concludes one iteration. After completing 40 times the main function will return the last generation that can be then processed to find the best individual.

4.3 Eclipse with PyDEV plugin

Eclipse is a very mature and extensible IDE (Integrated Development Environment). It is implemented mostly in Java language and was originally meant for Java development. Because of extensibility of the platform, support for other languages including Python was added.

PyDEV is an extension for Eclipse platform enabling Python language support. The feature list consists but is not limited to:

- syntax highlighting,
- wizards for creating new projects, modules, packages, etc.,
- unit testing integration,
- support for Python, Jython (Python implementation in Java) and IronPython (.NET implementation of Python)
- code completion, code navigation

- debugging
- reporting and code statistics
- code refactoring

Combination of Eclipse and PyDEV was chosen as development platform for this project mostly because of years of experience of the author using Eclipse based products.

5 Solution domain

In this section the details of the solution implemented are presented starting with an overview of prior work relating to the problem.

5.1 Prior work overview

A considerable amount of research was done in this field. Generally, there are two approaches to solving the problem. The first is using expert systems and the other is using optimisation methods (Rutherford, 2009). Expert systems were historically the first attempt at generating tablatures from music scores (Sayegh, 1989). Expert systems are rule based systems that mimic human experts in solving particular problem. Some findings using this method, especially construction of the fitness function inspired further researches that were using optimisation methods, most notably dynamic programming (Aleksander Radisavljevic, 2004) and genetic algorithms (Tuohy & Potter, 2006) (Rutherford, 2009), (D. R. Tuohy, 2005) to solve the problem. In all researches there are recurring themes such as high level approach, phrasal segmentation, musical complexity, performance features, musical style and algorithm training (Rutherford, 2009).

5.2 Implementation of the genetic algorithm

The following sections describe details of the genetic algorithm designed to solve the problem of converting classically notated music to guitar tablatures.

5.2.1 Source code structure

The source code is organized in four Python modules:

- core.py

  This module contains various functions implementing different parts of the genetic algorithm as well as some helper functions. These include implementation of initialization of the algorithm, mutation operator, generating all possible fingerings for given note, etc.

- TestCore.py

  This module contains a unit test suite for core.py module.

- gentab.py

  This is the main module where the genetic algorithm main function is implemented as well as reporting logging and visualisation of fitness function minimum values across generations.

- input.py

  This module contains manually translated input scores stored as Python variables. The variables were used to test the algorithm.

Different relations between the modules are presented in the following UML package diagram. The gentab.py is the main module that uses other packages to accomplish implementing the genetic algorithm. It is the gentab.py module that references DEAP framework.

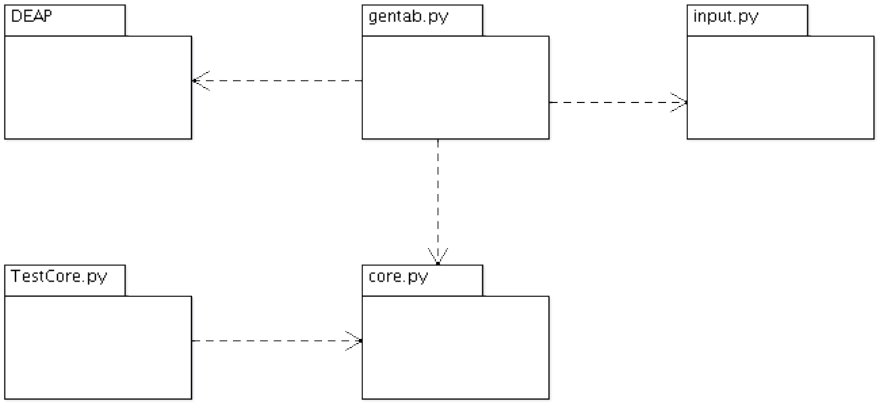


Fig 14: Package UML diagram of the implementation (author)

5.2.2 Encoding / Data structures

5.2.2.1 Classical music notation data structure

A piece of music written using classical notation is represented as a series of notes that can be read from left to right. This direction of reading represents time line. The rhythmic aspects of notation are not taken into consideration (whether a particular note is a full note, half note, quarter note, etc.). This is a justified choice since tablatures do not contain this information.

At Python data structures level, the classical music notation is represented as a list of lists of tuples (tuple is Python's data structure that is synonymous with an immutable list and is denoted using parentheses). Each such tuple is in the following form:

```
(a,b,c)
```

where **a** is the note pitch, ***b*** indicates accidentals ('#' for sharp, 'b' for flat, 'n' for natural) and ***c*** denotes octave number. For example the following structure represents C major scale played in order over two octaves:

```
cMajorScale = [[('C','n', 4)],
               [('D','n', 4)],
               [('E','n', 4)],
               [('F','n', 4)],
               [('G','n', 4)],
               [('A','n', 4)],
               [('B','n', 4)],
               [('C','n', 5)],
               [('D','n', 5)],
               [('E','n', 5)],
               [('F','n', 5)],
               [('G','n', 5)],
               [('A','n', 5)],
               [('B','n', 5)],
               [('C','n', 6)]
              ]
```

Example 11: Classical music notation representation

Structures of this form are used in the designed Genetic Algorithm to generate the initial population of solutions

5.2.2.2 Tablature data structure

The tablature data structure is very similar to the structure representing classical music notation. The top level list represents the whole tablature. The second level list represents one point in time at which one or more notes are played. Each note is represented by a tuple of two integers. The first integer designates a fret number which needs to be depressed to get the particular note, the second integer designates the string number which needs to be depressed to get the particular note. Strings are numbered from 0 to 5 which corresponds to the low E string to the high E string. For example the following example shows Python representation of a tablature and its notation equivalent:

```
tablature = [[(1,5)],
[(2,4)],
[(3,3)],
[(4,2)],
[(5,1)],
[(6,0)],
[(7,5), (7,4), (7,3)]
]
```

```
E---1-----------------7-
B------2--------------7-
G---------3-----------7-
D------------4----------
A---------------5-------
E------------------6----
```

Example 12: Tablature notation representation and its equivalent notation

The structure is used by the designed Genetic Algorithm to represent solutions.

5.2.3 Population initialization

DEAP comes with a number of built in helper functions that help to initialize population of solutions for the genetic algorithm to work on. However, these cover relatively simple representations like list of boolean, integer or floating point values. Since the representation chosen to hold tablature is slightly more complex, a custom initialization function is required. The initial population is generated off the input notation using a function that for each classical element of notation (note pitch) is able to generate one of corresponding tablature fingerings. An example of randomly generated tablature for C Major scale played over two octaves is presented below. Please note that this tablature is correct in that the notes are in fact two octave C Major scale notes that are ordered from the lowest to the highest pitch, however it is very impractical to play it the way it is arranged because of the unnaturally long jumps across the fretboard:

```
E---------------------------------1--3--------8-
B-----------------------------------------12---
G---------------0--------5----------------------
D------------------7--9-----12------------------
A---3--5-----8-----------------19-------24------
E---------12-------------------------------------
```

Example 13: Randomly generated tablature.

An additional complexity in generating random tablatures arises when chords need to be generated. This is because by simply choosing randomly each note of a chord, there could be a situation where two or more notes are fretted on the same string, which is impossible to play. The approach is to use Cartesian product of fingerings of all the notes in a given chord to determine the possible fingerings of chords (i.e., this sequences that don't have two or more notes fretted on the same string) and then one of the fingerings is chosen randomly. The implementation of the function is presented below.

```
def initTablature(container, classicalScore):
    '''
    initializes tablature from a classical notated
representation
    to a tablature with random fingerings
    '''
    result = container()

    for timeslot in classicalScore:
        tabTimeslot = list()

        #if there is one note in the slot then choose fingering
randomly
        if len(timeslot) == 1:
            fingerings = getAllFingerings(timeslot[0])
            #choose one fingering randomly
            tabTimeslot.append(choice(fingerings))
        else: #otherise things are tricky, as we cannot choose
two fingerings on the same string...
            #You can't play two notes on the same string!
            #first check if it's even possible to have
fingerings on different strings
            validFingerings =
getValidChordsForFingerings(timeslot)


            tabTimeslot = list(choice(validFingerings))

        result.append(tabTimeslot)

    return result
```

Example 14: Tablature initialization function.

5.2.4 Genetic operators

5.2.4.1 Crossover

DEAP comes with a set of helper functions implementing basic crossover operations. The algorithm reuses DEAP's implementation of two point crossover. This can be done without any modifications or post crossover integration checks since the representation of tablature is list based and swapping parts of tablature will result in a valid tablature corresponding to given piece of music.

5.2.4.2 Mutation

Although DEAP comes with a set of functions implementing different types of mutation operator, none of them can correctly handle the custom tablature representation. Hence, a custom mutation operator needs to be implemented. Let us recall that the representation of the tablature is in the form of list of lists of tuples. The top level list represents the whole piece of music. Each sublist represents a slot in time during which one or more notes may be required to be played. Each such note is represented by a tuple. The custom implementation of mutation operator iterates over the tablature representation and for each sublist (i.e. timeslot) changes fingering for a note with certain probability (mutation probability). If at the slot that is to be mutated there happens to be more than one note, the situation is slightly more complex. The algorithm cannot simply change one of the notes of the chord since it could result in a chord with two notes fretted at the same string, which is impossible to play, therefore the algorithm needs to take this into account so that a result of mutation is valid. The approach is to use Cartesian product of fingerings of all the notes in a given chord to determine the possible fingerings of chords (i.e., this sequences that don't have two or more notes fretted on the same string) and then one of the fingerings is chosen randomly. The mutation operator implementation is shown below.

```
def tabMutate(individual, indpb):
    '''
    performs mutation on the individual. Each chord in the
score will be altered with
    probability of indpb.
    '''
    for indx in xrange(len(individual)):
        if random() < indpb:

            timeSlotLength = len(individual[indx])

            allFingerings = None

            if timeSlotLength == 1:
                fingering = individual[indx][0]
                noteTuple =
convertFingeringTupleToNoteTuple(fingering)
                allFingerings = getAllFingerings(noteTuple)
            else:
                chordFingering = individual[indx]
                chordNotes = list()
                for fingering in chordFingering:

chordNotes.append(convertFingeringTupleToNoteTuple(fingering))
                allFingerings =
getValidChordsForFingerings(chordNotes)

            #if this is the only possible fingering, then
nothing to mutate
            if len(allFingerings) > 1 :
                #simulate do while loop
                doWhileCondition = True
                while doWhileCondition:
                    randomFingering = choice(allFingerings)
                    if timeSlotLength == 1:
                        if (randomFingering != fingering):
                            doWhileCondition = False
                            individual[indx][0] =
randomFingering
                    else:
                        if len(set(chordFingering) -
set(randomFingering)) != 0:
                            doWhileCondition = False
                            individual[indx] = randomFingering


    return individual,
```

Example 15: Tablature mutation operator.

5.2.5 Fitness function

The fitness function is quite problematic, because unlike in the simple examples of finding the minimum or maximum sum of numbers, in this case the function cannot be that simply defined. The aim of the algorithm is to find a good quality tablature. This term is in itself very subjective and to a certain extent a matter of preference of individual players. The function most definitely is a good candidate for composite function or multi criteria function. Luckily the DEAP framework provides support for such fitness evaluation. For this project the following criteria have been chosen based on the author's guitar playing experience to evaluate fitness:

- The average of fret positions should be minimized. This means that tablatures that position notes closer to the neck of the guitar will be preferred.
- The average distance between fretted notes should be minimized, this means that the notes should be as close together as possible.
- A chord cannot span more than four frets unless an open string is used. It is assumed that chords spanning over four frets are physically impossible to play. Such arrangements are considered not valid.

Each of the first two mentioned criteria correspond to a function that implements the evaluation logic. They are used during the main loop of the genetic algorithm. The last criterion is treated as a constrain on a valid tablature and is used by the initialization and mutation operator functions. The individual functions each implementing one criterion in the multi-objective fitness are shown below

```
def averageFretNumber(individual):
    '''
    for a given tablature returns an average of all fret
positions
    '''
    noteCounter = 0
    fretPositionSum = 0.0
    for timeSlot in individual:

        noteCounter += len(timeSlot)
        for note in timeSlot:
            fretPositionSum += note[0]
    #return average

    return 0 if noteCounter == 0 else fretPositionSum /
noteCounter


def averageSubsequentFretDistance(individual):
    '''
    for a given tablature returns an average distance between
subsequen frets positions
    '''
    timeSlotCounter = 0
    fretPositionDistanceSum = 0.0
    lastFretPosition = None
    for timeSlot in individual:

        #open string positins will not contribute to the average
        if timeSlot[0][0] != 0:
            #consider only first note
            if lastFretPosition == None:
                lastFretPosition = timeSlot[0][0]
            else:
                timeSlotCounter += 1
                fretPositionDistanceSum += abs(lastFretPosition
- timeSlot[0][0])
                lastFretPosition = timeSlot[0][0]

    #return average
    return 0 if timeSlotCounter == 0 else
fretPositionDistanceSum / timeSlotCounter
```

Example 16: Tablature fitness functions.

5.3 Testing the algorithm

Running the algorithm requires Python 2.6 interpreter with DEAP framework installed (version at least 0.8.2). Additionally, for visualisation of results **matplotlib** and **numpy** libraries should be present as well.

A suite of unit tests have been developed to test functionality of all implemented functions. These are located in the TestCore.py class.

5.3.1 Algorithm setup

The algorithm has been parametrized with the following functions and values:

Parameter	Value
Crossover probability	0.5
Mutation of individual probability	0.2
Mutation of a gene probability	0.05
Number of generations	50

5.3.2 Input Data

The algorithm was run using four music scores with increasing level of length and complexity. All scores can be found in the appendix section of this report. The first score was an ascending C Major scale over two octaves. The second score was the first four bars of guitar intro of Stairway to Heaven by Led Zeppelin. The third was one guitar part from a Duet in F, a piece of music from Volume 1 of A Modern Method For Guitar by William Leavitt. Finally as the most complex score used was Charlie Christian guitar solo from Flying Home tune transcribed by the author of this report.

Each score was manually translated into the algorithm representation of music score and stored in a Python source file (input.py).

5.3.3 Statistics of the evolution algorithm

For each of the input scores used, the algorithm gathered statistics of the evolution process. These are presented below in both textual and graphic form. Whenever fitness value is presented as two dimensional vector, the first element corresponds to average fret position and the second to average subsequent fret distance.

5.3.3.1 C major scale

Running time of the algorithm: **233.84 sec**

Statistics of the fitness function across generations:

Generation	Number of individuals evaluated	Standard deviation	Maximum	Average	Minimum
0	300	[1.5747, 1.3209]	[14.9333, 11.0769]	[10.0976, 6.4559]	[5.6667, 2.7500]
1	172	[1.5804, 1.3651]	[13.6667, 10.2500]	[10.0880, 6.4393]	[5.6667, 2.7500]
2	170	[1.6059, 1.3386]	[14.2667, 10.2500]	[10.0858, 6.4659]	[5.6667, 2.8333]
3	173	[1.6342, 1.3523]	[14.3333, 11.0769]	[10.0891, 6.4785]	[4.7333, 3.0909]
4	196	[1.6784, 1.4043]	[14.3333, 11.0769]	[10.0853, 6.4643]	[4.7333, 2.4286]
5	170	[1.6454, 1.4163]	[14.3333, 10.5833]	[10.0956, 6.4292]	[4.8000, 2.9000]
6	185	[1.7188, 1.5061]	[14.0000, 11.1538]	[10.1013, 6.4258]	[4.8000, 2.9000]
7	176	[1.7408, 1.5033]	[14.0000, 11.1538]	[10.1136, 6.4406]	[4.8000, 2.9000]
8	190	[1.7910, 1.4572]	[14.0000, 11.2143]	[10.0840, 6.4119]	[4.8000, 2.9000]
9	182	[1.8240, 1.4891]	[14.6000, 11.2143]	[10.0838, 6.3974]	[3.8667, 2.4167]
10	165	[1.8732, 1.5246]	[14.6000, 11.2857]	[10.0838, 6.3811]	[4.2000, 2.2308]
11	158	[1.8668, 1.5347]	[14.6000, 11.6154]	[10.0767, 6.4322]	[4.2000, 2.2308]
12	192	[1.9010, 1.5619]	[14.6000, 11.6154]	[10.0893, 6.4501]	[3.8667, 2.4545]
13	187	[1.9157, 1.6345]	[14.6000, 11.6154]	[10.1204, 6.4723]	[3.8667, 2.4545]
14	184	[1.9357, 1.6926]	[14.6000, 11.6154]	[10.1338, 6.4817]	[3.8667, 2.4545]
15	173	[1.9504, 1.6985]	[14.7333, 11.7692]	[10.1569, 6.5099]	[4.2000, 2.6923]
16	184	[1.9671, 1.6735]	[14.7333, 11.7692]	[10.1622, 6.4813]	[4.4667, 2.1818]
17	176	[1.9770, 1.6772]	[14.4000, 11.7692]	[10.1647, 6.4893]	[4.2000, 2.7692]
18	181	[2.0012, 1.6734]	[14.6000, 11.7692]	[10.1667, 6.4770]	[4.2000, 3.0769]
19	162	[1.9514, 1.6358]	[14.6667, 11.7692]	[10.1902, 6.5167]	[4.4667, 3.3636]
20	173	[1.9794, 1.6827]	[15.3333, 12.3846]	[10.2033, 6.4947]	[4.1333, 2.4545]
21	189	[1.9549, 1.6523]	[15.0000, 12.3846]	[10.2036, 6.4730]	[4.2000, 2.9000]
22	170	[1.9531, 1.6718]	[15.0000, 11.5000]	[10.2073, 6.4400]	[4.2000, 2.9000]
23	174	[1.9550, 1.6750]	[15.0000, 11.5000]	[10.2147, 6.4102]	[4.5333, 2.9286]
24	167	[1.9498, 1.6433]	[14.6667, 11.5000]	[10.2129, 6.4004]	[4.5333, 2.8462]
25	177	[1.9762, 1.6170]	[14.6667, 11.5000]	[10.2016, 6.4540]	[4.5333, 2.6364]
26	154	[1.9901, 1.6472]	[15.0000, 11.5000]	[10.1938, 6.4563]	[4.5333, 2.6364]
27	180	[2.0158, 1.6902]	[14.6667, 10.9286]	[10.1842, 6.4290]	[4.5333, 2.6364]
28	189	[1.9901, 1.7115]	[15.0000, 12.0000]	[10.1787, 6.4261]	[4.2000, 2.8462]

29	194	[1.9985, 1.7375]	[15.0000, 12.0000]	[10.1698, 6.4236]	[4.2000, 2.5385]
30	175	[2.0114, 1.6723]	[15.0000, 11.6667]	[10.1773, 6.3705]	[4.2000, 2.5385]
31	192	[1.9847, 1.6918]	[15.3333, 11.6667]	[10.1787, 6.3715]	[4.2000, 2.0000]
32	184	[1.9790, 1.7262]	[15.0000, 12.0000]	[10.1873, 6.3317]	[4.2000, 2.0000]
33	187	[2.0063, 1.7221]	[15.0000, 11.0769]	[10.1847, 6.2855]	[4.5333, 1.6923]
34	165	[2.0049, 1.7108]	[15.0000, 10.8182]	[10.1691, 6.2719]	[4.5333, 1.8333]
35	190	[2.0181, 1.6759]	[15.0000, 10.8182]	[10.1907, 6.2532]	[4.5333, 1.8333]
36	184	[2.0240, 1.6628]	[15.0000, 10.8182]	[10.1867, 6.2646]	[3.8667, 1.8333]
37	181	[2.0412, 1.6694]	[15.0000, 10.8182]	[10.1776, 6.2622]	[3.8667, 1.8333]
38	194	[2.0295, 1.7158]	[15.0000, 11.0714]	[10.1662, 6.2639]	[3.8667, 2.0000]
39	198	[1.9956, 1.6898]	[14.0667, 10.6429]	[10.1849, 6.2448]	[4.4000, 2.5000]
40	165	[2.0205, 1.7582]	[14.7333, 11.4615]	[10.2044, 6.2503]	[4.0667, 2.5000]
41	179	[2.0316, 1.7795]	[14.9333, 11.7857]	[10.2227, 6.2884]	[4.0667, 1.8182]
42	161	[2.0371, 1.7487]	[16.0000, 11.7857]	[10.2100, 6.2023]	[4.0667, 1.7692]
43	187	[2.0451, 1.6941]	[16.0000, 12.6923]	[10.2187, 6.2124]	[3.7333, 2.0833]
44	175	[2.0093, 1.6696]	[15.3333, 12.6923]	[10.2122, 6.1909]	[3.7333, 2.0833]
45	184	[1.9618, 1.6713]	[15.3333, 11.8333]	[10.2096, 6.1797]	[3.7333, 2.0833]
46	171	[1.9688, 1.7372]	[14.4000, 11.8333]	[10.2178, 6.2126]	[3.7333, 2.0833]
47	176	[1.9093, 1.7666]	[14.4000, 11.8333]	[10.2011, 6.2391]	[3.7333, 2.0833]
48	197	[1.9390, 1.7468]	[15.2667, 11.8333]	[10.1771, 6.2363]	[3.7333, 2.0833]
49	175	[1.9532, 1.6992]	[14.7333, 10.6667]	[10.1809, 6.2365]	[4.0000, 2.0000]
50	204	[1.9866, 1.6308]	[14.7333, 11.3571]	[10.1920, 6.2345]	[4.3333, 2.3846]

Minimum fitness plot of across generations

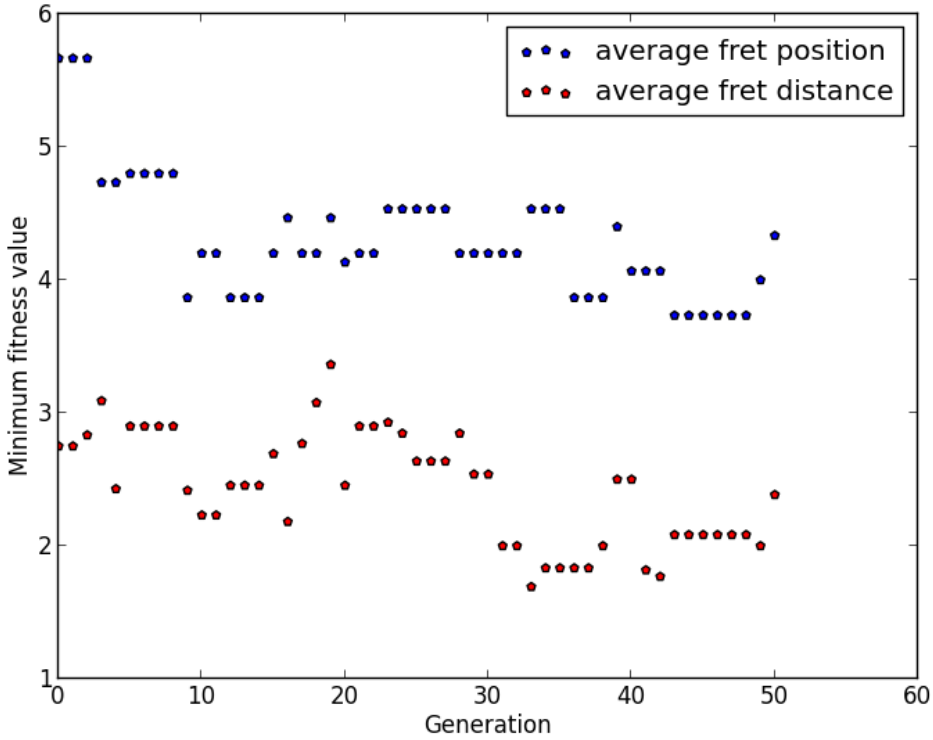


Fig 15: Minimum fitness plot for C Major score

The best individuals fitness values

Individual number	Average fret position	Average fret distance
1	3.7333333333333334	2.8181818181818183
2	3.8666666666666667	2.4166666666666665
3	4.2000000000000002	2.2307692307692308
4	4.333333333333333	2.0833333333333335
5	4.666666666666667	2.0
6	5.0	1.8333333333333333
7	6.666666666666667	1.8181818181818181
8	7.333333333333333	1.6923076923076923

5.3.3.2 Stairway to Heaven

Running time of the algorithm: **257.88 sec**

Statistics of the fitness function across generations:

Generation	Number of individuals evaluated	Standard deviation	Maximum	Average	Minimum
0	300	[0.9972, 1.1419]	[11.8611, 10.3182]	[8.8675, 7.0125]	[5.8889, 4.1364]
1	175	[1.0322, 1.1733]	[11.8611, 10.4286]	[8.8625, 6.9739]	[5.8889, 4.1364]
2	184	[1.0701, 1.2152]	[11.6111, 10.3182]	[8.8605, 6.9752]	[5.6111, 3.5217]
3	183	[1.1179, 1.2309]	[11.9722, 11.3810]	[8.8607, 6.9547]	[5.6111, 3.5217]
4	183	[1.1363, 1.2942]	[11.6667, 11.3810]	[8.8680, 6.9661]	[5.7500, 3.5217]
5	181	[1.1574, 1.3288]	[11.9722, 11.3810]	[8.8560, 6.9558]	[5.6667, 3.5217]
6	148	[1.1604, 1.3505]	[11.8889, 10.9524]	[8.8507, 6.9827]	[5.6667, 3.6667]
7	175	[1.1983, 1.3606]	[12.2222, 10.7826]	[8.8458, 6.9694]	[5.6667, 3.7600]
8	163	[1.2278, 1.3777]	[11.6944, 10.7826]	[8.8444, 6.9412]	[5.6667, 3.7727]
9	185	[1.2259, 1.4094]	[11.6944, 10.5417]	[8.8471, 6.9697]	[4.8889, 3.7727]
10	181	[1.2320, 1.4319]	[11.6389, 10.6087]	[8.8513, 6.9507]	[5.5556, 3.5714]
11	175	[1.2598, 1.4656]	[11.6667, 10.7083]	[8.8523, 6.9702]	[5.5556, 3.5714]
12	168	[1.2493, 1.4661]	[11.7500, 11.1250]	[8.8531, 6.9796]	[5.5556, 3.5714]
13	189	[1.2465, 1.4986]	[11.7778, 12.0000]	[8.8550, 7.0022]	[5.5556, 3.3500]
14	188	[1.2528, 1.4698]	[11.8056, 12.0000]	[8.8456, 6.9675]	[5.5556, 3.3500]
15	190	[1.2825, 1.4856]	[11.6667, 12.0000]	[8.8507, 6.9668]	[5.5556, 3.3500]
16	195	[1.2944, 1.5357]	[12.1111, 11.8750]	[8.8496, 6.9999]	[5.5556, 3.3500]
17	199	[1.2845, 1.5579]	[11.8333, 11.8750]	[8.8408, 6.9754]	[5.5556, 3.3500]
18	177	[1.2667, 1.5457]	[12.2500, 11.5652]	[8.8416, 6.9657]	[5.5556, 3.5714]
19	182	[1.2844, 1.5811]	[11.9722, 11.9167]	[8.8312, 6.9379]	[5.3056, 3.5000]
20	173	[1.2828, 1.5756]	[11.5833, 11.9167]	[8.8261, 6.9250]	[4.6111, 3.2609]
21	187	[1.2755, 1.5888]	[11.8611, 11.9167]	[8.8154, 6.9225]	[4.6111, 3.3182]
22	193	[1.2735, 1.5908]	[11.8611, 11.9167]	[8.8281, 6.9233]	[5.2778, 3.0455]
23	180	[1.2549, 1.5771]	[11.8611, 11.9167]	[8.8377, 6.9055]	[5.7778, 3.6087]
24	183	[1.2500, 1.5978]	[12.5278, 11.8333]	[8.8468, 6.8947]	[5.5556, 3.7727]
25	177	[1.2864, 1.5975]	[12.2500, 11.8333]	[8.8431, 6.8641]	[5.5556, 3.5909]
26	176	[1.2774, 1.6066]	[12.6667, 11.5833]	[8.8520, 6.9017]	[5.6667, 3.4167]
27	183	[1.2630, 1.5893]	[12.0556, 11.2174]	[8.8464, 6.9245]	[5.7778, 3.5000]
28	167	[1.2608, 1.5934]	[11.9444, 11.2174]	[8.8421, 6.9047]	[5.5556, 3.5000]
29	174	[1.2830, 1.5640]	[12.6111, 11.5833]	[8.8548, 6.8733]	[5.5556, 3.5833]
30	183	[1.2832, 1.5622]	[12.1667, 11.6667]	[8.8556, 6.8594]	[5.6111, 3.4762]
31	169	[1.3036, 1.5651]	[12.4444, 11.7273]	[8.8468, 6.8686]	[4.7500, 3.6087]
32	177	[1.3238, 1.5856]	[12.4444, 11.5000]	[8.8438, 6.8762]	[5.1667, 3.5217]
33	175	[1.3234, 1.6087]	[12.4444, 12.0000]	[8.8389, 6.9074]	[4.8611, 3.4091]
34	181	[1.3394, 1.5800]	[12.4444, 11.6522]	[8.8382, 6.9428]	[5.2778, 3.1818]
35	169	[1.3299, 1.5902]	[12.4444, 11.5833]	[8.8354, 6.9144]	[5.0278, 3.1818]
36	174	[1.3181, 1.5910]	[12.0833, 11.1905]	[8.8308, 6.8952]	[4.7500, 3.1818]
37	188	[1.3358, 1.6023]	[12.3333, 11.1304]	[8.8224, 6.8983]	[5.1389, 2.9545]
38	194	[1.3758, 1.6063]	[12.3333, 11.3182]	[8.8269, 6.8730]	[4.9444, 2.9545]
39	175	[1.4027, 1.6134]	[12.3333, 11.1304]	[8.8256, 6.8578]	[4.9444, 2.4783]
40	177	[1.4218, 1.6457]	[12.3333, 12.0909]	[8.8150, 6.8456]	[4.9444, 2.4783]
41	176	[1.4066, 1.6434]	[12.7500, 12.0909]	[8.8169, 6.8645]	[5.1944, 2.4783]
42	187	[1.4389, 1.6888]	[12.7500, 12.9091]	[8.8238, 6.8859]	[4.2778, 2.9524]

43	180	[1.4507, 1.6849]	[12.7500, 12.9091]	[8.8238, 6.8957]	[4.3889, 3.2174]
44	186	[1.4559, 1.6870]	[12.4444, 12.9091]	[8.8340, 6.8863]	[4.2500, 2.6667]
45	189	[1.4550, 1.7484]	[12.5000, 12.9091]	[8.8326, 6.9152]	[4.3611, 2.6667]
46	188	[1.4626, 1.7842]	[12.0833, 12.0909]	[8.8428, 6.8983]	[4.3611, 2.6667]
47	170	[1.4523, 1.7780]	[12.0278, 12.2400]	[8.8353, 6.9197]	[4.0833, 2.3810]
48	178	[1.4590, 1.7826]	[12.1667, 12.0909]	[8.8483, 6.9118]	[4.8889, 2.5238]
49	180	[1.4680, 1.7981]	[12.1667, 12.1364]	[8.8528, 6.9065]	[4.8889, 2.5238]
50	165	[1.4700, 1.7747]	[12.4444, 13.3333]	[8.8458, 6.8994]	[4.8889, 2.4091]

Minimum fitness plot of across generations

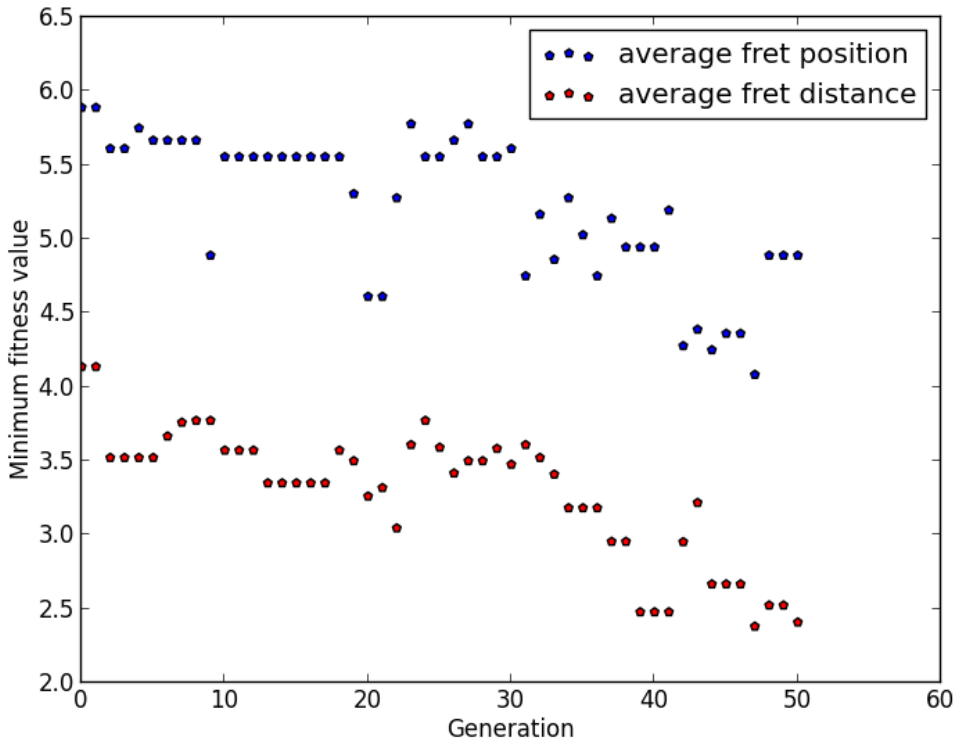


Fig 16: Minimum fitness plot for Stairway To Heaven score

The best individuals fitness values

Individual number	Average fret position	Average fret distance
1	4.083333333333333	2.7619047619047619
2	4.25	2.6666666666666665
3	5.3888888888888893	2.5238095238095237
4	5.666666666666667	2.4782608695652173

Individual number	Average fret position	Average fret distance
5	5.7777777777777777	2.4090909090909092
6	6.1944444444444446	2.3809523809523809

5.3.3.3 Duet in F

Running time of the algorithm: **385.72 sec**

Statistics of the fitness function across generations:

Generation	Number of individuals evaluated	Standard deviation	Maximum	Average	Minimum
0	300	[0.4864, 0.4315]	[9.1377, 7.0270]	[7.8886, 5.8205]	[6.4275, 4.7193]
1	158	[0.4925, 0.4491]	[9.1087, 7.1892]	[7.8882, 5.8172]	[6.2899, 4.6875]
2	164	[0.5133, 0.4723]	[9.0725, 7.1339]	[7.8816, 5.8170]	[6.4420, 4.5766]
3	172	[0.5068, 0.4802]	[9.5507, 7.0090]	[7.8790, 5.8231]	[6.5145, 4.4561]
4	179	[0.5288, 0.4939]	[9.5507, 7.0090]	[7.8870, 5.8314]	[6.4275, 4.4561]
5	186	[0.5437, 0.5002]	[9.5870, 7.3704]	[7.8844, 5.8378]	[6.4058, 4.2696]
6	166	[0.5642, 0.5160]	[9.5435, 7.3704]	[7.8851, 5.8369]	[6.3986, 4.2696]
7	192	[0.5673, 0.5258]	[9.4275, 7.2566]	[7.8774, 5.8356]	[6.0580, 4.2696]
8	183	[0.5763, 0.5287]	[9.4348, 7.2679]	[7.8679, 5.8310]	[6.0580, 4.5044]
9	180	[0.5856, 0.5285]	[9.5797, 7.2679]	[7.8632, 5.8226]	[6.0942, 4.5495]
10	185	[0.5837, 0.5338]	[9.3623, 7.3694]	[7.8550, 5.8138]	[6.0942, 4.5929]
11	181	[0.5951, 0.5475]	[9.3623, 7.3694]	[7.8519, 5.8146]	[6.0942, 4.5088]
12	176	[0.5963, 0.5541]	[9.3623, 7.3694]	[7.8492, 5.8086]	[6.0942, 4.5088]
13	183	[0.6105, 0.5418]	[9.3261, 7.3694]	[7.8546, 5.8138]	[6.2319, 4.1593]
14	176	[0.6288, 0.5438]	[9.4928, 7.3694]	[7.8515, 5.8174]	[5.7681, 4.1593]
15	189	[0.6440, 0.5500]	[9.4203, 7.2982]	[7.8558, 5.8123]	[5.7681, 4.3894]
16	178	[0.6475, 0.5631]	[9.4928, 7.1858]	[7.8572, 5.8104]	[5.9855, 4.4464]
17	154	[0.6492, 0.5696]	[9.4928, 7.1545]	[7.8619, 5.8108]	[5.9855, 4.4464]
18	182	[0.6552, 0.5811]	[9.3768, 7.2281]	[7.8605, 5.8127]	[5.8768, 4.3009]
19	182	[0.6593, 0.5898]	[9.4638, 7.2909]	[7.8615, 5.8148]	[5.8768, 4.0982]
20	145	[0.6643, 0.5820]	[9.5942, 7.3243]	[7.8548, 5.8201]	[5.8768, 4.0091]
21	177	[0.6646, 0.5940]	[9.5580, 7.4865]	[7.8558, 5.8171]	[5.6522, 4.0091]
22	184	[0.6684, 0.5964]	[9.7536, 7.4865]	[7.8625, 5.8120]	[5.8986, 4.0091]
23	181	[0.6767, 0.6029]	[9.7899, 7.6036]	[7.8588, 5.8106]	[5.6884, 3.9727]
24	179	[0.6693, 0.5964]	[9.7536, 7.5321]	[7.8620, 5.8265]	[5.8261, 4.1892]
25	176	[0.6616, 0.5925]	[9.5145, 7.5138]	[7.8615, 5.8185]	[6.1232, 4.1892]
26	182	[0.6604, 0.5987]	[9.5072, 7.5138]	[7.8561, 5.8200]	[5.9203, 4.1892]
27	182	[0.6734, 0.6086]	[9.5072, 7.5175]	[7.8562, 5.8241]	[6.0145, 4.1892]
28	166	[0.6745, 0.6236]	[9.5072, 7.5175]	[7.8558, 5.8348]	[6.1957, 4.1892]
29	188	[0.6667, 0.6291]	[9.5000, 7.5175]	[7.8566, 5.8259]	[6.1812, 4.0631]

30	187	[0.6656, 0.6248]	[9.5072, 7.5909]	[7.8614, 5.8205]	[6.1232, 4.0631]
31	187	[0.6732, 0.6237]	[9.5072, 7.5909]	[7.8646, 5.8272]	[6.1232, 4.3451]
32	191	[0.6839, 0.6170]	[9.5072, 7.5625]	[7.8627, 5.8238]	[6.1232, 4.2500]
33	179	[0.6841, 0.6164]	[9.5072, 7.5625]	[7.8637, 5.8206]	[6.1232, 4.2500]
34	198	[0.6955, 0.6158]	[9.5072, 7.5625]	[7.8682, 5.8283]	[6.1232, 4.4505]
35	176	[0.7013, 0.6136]	[9.5072, 7.5625]	[7.8711, 5.8298]	[5.8913, 4.2162]
36	183	[0.6954, 0.5997]	[9.5725, 7.4545]	[7.8676, 5.8277]	[5.8913, 4.2182]
37	176	[0.6971, 0.5903]	[9.4565, 7.4224]	[7.8661, 5.8201]	[5.9928, 4.3423]
38	180	[0.7003, 0.5913]	[9.4565, 7.4224]	[7.8736, 5.8224]	[5.9203, 4.3784]
39	181	[0.6969, 0.5980]	[9.3841, 7.2703]	[7.8708, 5.8211]	[5.9203, 4.3784]
40	176	[0.6901, 0.6029]	[9.4638, 7.4336]	[7.8751, 5.8293]	[5.7971, 4.2478]
41	179	[0.6892, 0.6048]	[9.4710, 7.4336]	[7.8821, 5.8359]	[5.7971, 4.2478]
42	180	[0.6801, 0.6124]	[9.4710, 7.4336]	[7.8845, 5.8393]	[5.7971, 4.1770]
43	176	[0.6795, 0.6187]	[9.6159, 7.7568]	[7.8842, 5.8379]	[5.9348, 4.2895]
44	161	[0.6894, 0.6225]	[9.5145, 7.7568]	[7.8762, 5.8273]	[5.9348, 4.2162]
45	161	[0.6870, 0.6156]	[9.5145, 7.4107]	[7.8797, 5.8245]	[5.8261, 4.0439]
46	164	[0.6912, 0.6081]	[9.5072, 7.3540]	[7.8803, 5.8312]	[5.8623, 4.3628]
47	185	[0.7057, 0.6167]	[9.5145, 7.7321]	[7.8754, 5.8276]	[5.8623, 4.3628]
48	198	[0.7140, 0.6214]	[9.5507, 7.6847]	[7.8783, 5.8279]	[5.8623, 4.1217]
49	181	[0.7204, 0.6167]	[9.5580, 7.5179]	[7.8752, 5.8214]	[5.8623, 4.0088]
50	162	[0.7176, 0.6215]	[9.7246, 7.4336]	[7.8688, 5.8293]	[5.9928, 4.0614]

Minimum fitness plot of across generations

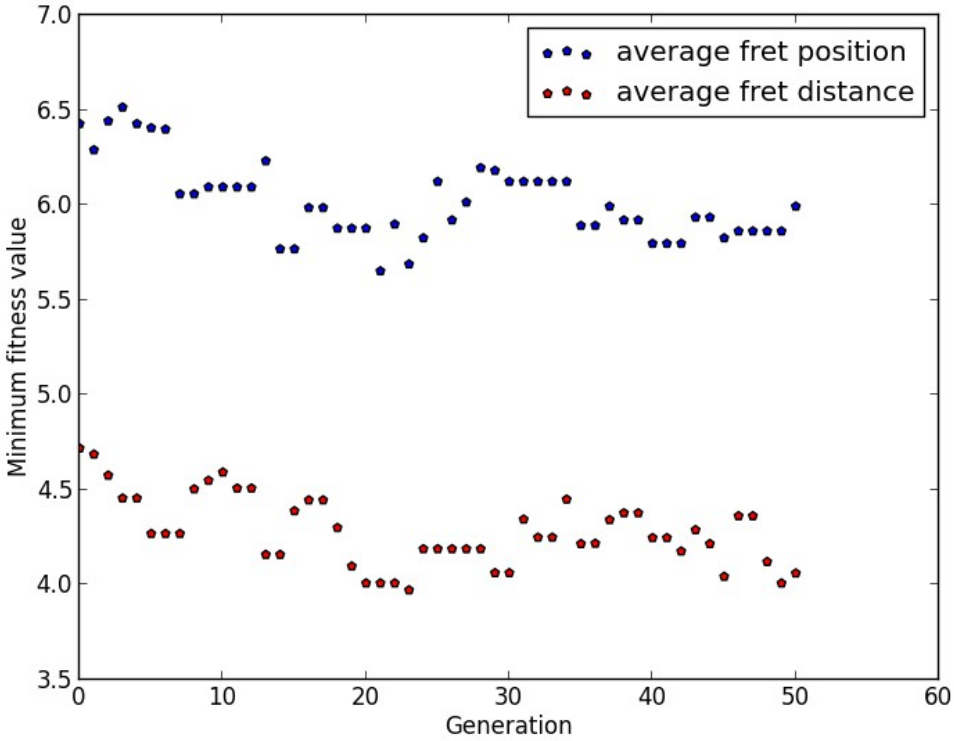


Fig 17: Minimum fitness plot for Duet in F score

The best individuals fitness values

Individual number	Average fret position	Average fret distance
1	5.6521739130434785	5.0272727272727273
2	5.7971014492753623	4.6759259259259256
3	5.86231884057971	4.5357142857142856
4	5.8913043478260869	4.3090909090909095
5	5.9420289855072461	4.3008849557522124
6	5.9927536231884062	4.1559633027522933
7	6.2608695652173916	3.9727272727272727

5.3.3.4 Flying Home

Running time of the algorithm: **406.14 sec**

Statistics of the fitness function across generations:

Generation	Number of individuals evaluated	Standard deviation	Maximum	Average	Minimum
0	300	[0.4916, 0.4296]	[13.7595, 8.2792]	[12.5059, 7.1283]	[11.3038, 5.9935]
1	174	[0.5073, 0.4443]	[13.9430, 8.1623]	[12.4994, 7.1166]	[10.8418, 5.9351]
2	190	[0.5360, 0.4657]	[14.0696, 8.4870]	[12.5003, 7.1153]	[10.9051, 5.6667]
3	171	[0.5489, 0.4881]	[13.9557, 8.3529]	[12.4995, 7.1050]	[10.9051, 5.6667]
4	177	[0.5457, 0.4959]	[13.9557, 8.4545]	[12.4964, 7.1120]	[10.9873, 5.5000]
5	197	[0.5764, 0.5218]	[13.9747, 8.6753]	[12.4957, 7.1150]	[11.0127, 5.6928]
6	167	[0.5952, 0.5236]	[14.0633, 8.9091]	[12.4970, 7.1130]	[10.6266, 5.8636]
7	153	[0.5985, 0.5287]	[13.9557, 8.9091]	[12.5006, 7.1056]	[10.6266, 5.8636]
8	190	[0.6041, 0.5324]	[14.1266, 8.7792]	[12.4988, 7.1030]	[10.6266, 5.8636]
9	197	[0.6119, 0.5434]	[14.1139, 8.7792]	[12.4949, 7.1063]	[10.8987, 5.6169]
10	183	[0.6169, 0.5644]	[13.9684, 8.9935]	[12.4977, 7.0997]	[10.9810, 5.6169]
11	177	[0.6030, 0.5700]	[13.9684, 8.9091]	[12.5021, 7.1192]	[10.7089, 5.6169]
12	178	[0.6039, 0.5701]	[13.9684, 9.3506]	[12.5060, 7.1209]	[10.6582, 5.6169]
13	183	[0.6118, 0.5780]	[14.0127, 9.3506]	[12.5098, 7.1252]	[11.0253, 5.4221]
14	180	[0.6317, 0.5879]	[14.0127, 9.3506]	[12.5114, 7.1184]	[11.0253, 5.4221]
15	169	[0.6293, 0.5940]	[14.4430, 9.1558]	[12.5132, 7.1195]	[10.8291, 5.6558]
16	184	[0.6475, 0.5901]	[14.4430, 9.2857]	[12.5135, 7.1172]	[10.8671, 5.5294]
17	175	[0.6506, 0.6079]	[14.0823, 9.2857]	[12.5117, 7.1219]	[10.4304, 5.5584]
18	179	[0.6470, 0.6012]	[13.9684, 8.9870]	[12.5140, 7.1315]	[10.4304, 5.5033]
19	202	[0.6411, 0.5997]	[14.0127, 8.8235]	[12.5067, 7.1311]	[10.4304, 5.3684]
20	179	[0.6459, 0.6074]	[14.5190, 9.1299]	[12.5005, 7.1313]	[10.4304, 5.3684]
21	164	[0.6593, 0.6041]	[14.5190, 9.0260]	[12.5011, 7.1377]	[10.4241, 5.3684]
22	169	[0.6708, 0.6003]	[14.5759, 9.0260]	[12.5056, 7.1339]	[10.4051, 5.6974]
23	184	[0.6810, 0.6007]	[14.5190, 8.8497]	[12.5146, 7.1359]	[10.4051, 5.6818]
24	192	[0.6874, 0.5943]	[14.3924, 8.9091]	[12.5156, 7.1364]	[10.6329, 5.3312]
25	161	[0.6861, 0.5927]	[14.3924, 8.9091]	[12.5147, 7.1471]	[10.9114, 5.4091]
26	186	[0.6956, 0.5920]	[14.3354, 9.1429]	[12.5189, 7.1482]	[10.9620, 5.5000]
27	177	[0.6910, 0.6011]	[14.7025, 8.8571]	[12.5156, 7.1475]	[10.6582, 5.6601]
28	182	[0.7021, 0.6097]	[14.5253, 9.1818]	[12.5131, 7.1476]	[10.1962, 5.5390]
29	171	[0.7061, 0.6120]	[14.3101, 8.8627]	[12.5120, 7.1538]	[10.5886, 5.1169]
30	185	[0.7066, 0.6271]	[14.3101, 9.0390]	[12.5081, 7.1527]	[10.5886, 5.5390]
31	181	[0.7090, 0.6231]	[14.5253, 9.0390]	[12.5058, 7.1534]	[10.5886, 5.5519]
32	177	[0.7111, 0.6338]	[14.3734, 9.0390]	[12.5102, 7.1599]	[10.7468, 5.5098]
33	179	[0.7134, 0.6436]	[14.3734, 9.0390]	[12.5094, 7.1531]	[10.6266, 5.5519]
34	193	[0.7062, 0.6510]	[14.3734, 9.0390]	[12.5077, 7.1468]	[10.6076, 5.4870]
35	196	[0.6920, 0.6579]	[14.1456, 9.0390]	[12.5131, 7.1460]	[10.2025, 5.4870]
36	174	[0.7020, 0.6590]	[14.3608, 9.1039]	[12.5097, 7.1438]	[10.2025, 5.0844]
37	173	[0.7126, 0.6606]	[14.7658, 9.1429]	[12.5099, 7.1464]	[10.2025, 5.0844]
38	169	[0.7112, 0.6684]	[14.5443, 9.0000]	[12.5124, 7.1503]	[10.2025, 5.2157]
39	182	[0.7100, 0.6743]	[14.5443, 8.9351]	[12.5123, 7.1498]	[10.2025, 5.1558]
40	169	[0.7085, 0.6888]	[14.2215, 8.9351]	[12.5084, 7.1527]	[10.2025, 5.0065]

41	161	[0.7212, 0.6893]	[14.2975, 9.1364]	[12.5064, 7.1441]	[10.3797, 4.7468]
42	173	[0.7186, 0.6741]	[14.1835, 8.8052]	[12.5008, 7.1491]	[10.4810, 4.9156]
43	148	[0.7159, 0.6760]	[14.2785, 8.7338]	[12.4973, 7.1528]	[10.2025, 4.8506]
44	187	[0.7239, 0.6704]	[14.1456, 8.7647]	[12.4923, 7.1560]	[10.4241, 5.0844]
45	181	[0.7395, 0.6713]	[14.3165, 8.7647]	[12.5014, 7.1615]	[10.4241, 5.0844]
46	185	[0.7405, 0.6691]	[14.4620, 8.7647]	[12.4954, 7.1630]	[10.4241, 5.0844]
47	188	[0.7399, 0.6693]	[14.4620, 8.7582]	[12.4949, 7.1676]	[10.4810, 5.0844]
48	154	[0.7401, 0.6723]	[14.2152, 8.7451]	[12.4974, 7.1676]	[10.4810, 5.0844]
49	177	[0.7478, 0.6656]	[14.2975, 8.7451]	[12.4894, 7.1747]	[10.4430, 5.1948]
50	189	[0.7480, 0.6733]	[14.2848, 8.7987]	[12.4893, 7.1751]	[10.4241, 5.1948]

Minimum fitness plot of across generations

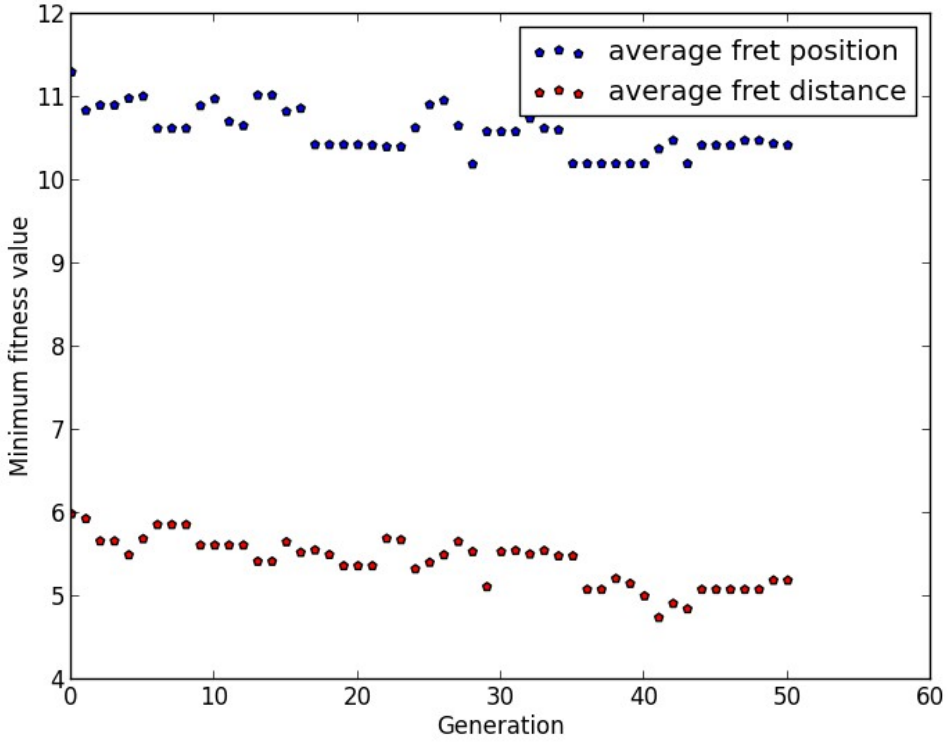


Fig 18: Minimum fitness plot for Flying Home score

The best individuals fitness values

Individual number	Average fret position	Average fret distance
1	10.19620253164557	5.7712418300653594
2	10.49367088607595	5.3071895424836599

Individual number	Average fret position	Average fret distance
3	10.670886075949367	4.9155844155844157
4	10.772151898734178	4.7467532467532472

5.3.4 Verifying results

Each generated tablature by the algorithm can be found in the appendix section of this report. Each of them was evaluated by the author by trying to perform them with paying attention to any unnatural arrangements sections that could be improved.

In general, the two first scores were quite adequately converted into guitar tablatures. The arrangements were quite easy to play. Unfortunately the same cannot be said about the two more complex scores. Though overall the quality was not too bad, there were sections of arrangements with unnatural jumps between string, or long jumps on the fretboard. Sometimes easier choices of fingerings were very obvious. The problem seems to be in the fitness evaluation function that doesn't capture more complex scenarios adequately.

6 Conclusion

This section summarizes what was achieved in this project with a focus on project objectives and whether they were met. Some recommendations for further work is given as well as some reflections on personal learning outcomes.

6.1 Summary of the project

The project met all of its main objectives to some degree. The details broken by objective are detailed in the following sections.

6.1.1 Implementing genetic algorithm using DEAP, genetic algorithms framework in Python

The objective of implementing the GA converting classically notated scores to guitar tablatures was fully met. The code was thoroughly tested and run many times over the course of this project and all defects that were found were removed. The implementation is not very performant and it certainly can be improved. The ideas

and details are described in the *Future development guidelines* section.

6.1.2 Finding appropriate representation of standard music notation and guitar tablature in the context of genetic algorithms.

The designed data structures for representing both guitar tablatures and classically notated music are based on nested sequences. They proved to be adequate to use in the GA context to a degree where a built-in crossover operator implementation from the DEAP framework could be reused without any modifications. The additional benefit of the data structures were that they are human readable and quite easy to process.

6.1.3 Finding appropriate fitness function to evaluate quality of a tablature

This objective was somehow met. The fitness function designed proved to be adequate for short and simple pieces of music. In more complex and long scores, the arrangements produced were of a lower quality, with some sections being very awkward to play.

6.1.4 Finding appropriate benchmark for generated solutions

This objective was somehow met. The solutions were evaluated by the author of this project. As was stated before, quality of tablature arrangements can be very subjective and be perceived differently by players with different preferences, so evaluation carried out as a part of this project is one man's opinion. Taking into consideration that the man is also the author of the project there is a big chance that the opinion is biased. One way of improving this situation would be to release the produced tablatures to wider audience of players and collate their opinions on the quality of the arrangements.

6.2 Future development guidelines

There are many aspects of the solution that can be improved on. These are listed and explained below.

6.2.1 Fitness evaluation

The fitness function that evaluates the solutions during the evolution process turned out to be too simplistic to adequately weed out bad arrangements. During verification phase of the project, it was found that some section of produced arrangements were unnecessarily difficult to play. The designed fitness function did quite good job in that simple scores were accurately translated, but more complex scores need a refined fitness function to produce higher quality arrangements.

6.2.2 Performance

The algorithm for the larger scores took few minutes to complete. This is not acceptable performance for end-user application. There are several ways in which performance of the algorithm could be improved. First of all DEAP comes with a distribution module that allows for distributing the work of the algorithm between several CPUs to improve performance. Secondly, the bottle neck of the algorithm are the mutation operation and fitness evaluation implementations. These could be reimplemented in C for better performance (this is a common practice in Python environment). And finally the PyPy Python interpreter with JIT (Just-In-Time) compiler could be used to speed up execution of the algorithm.

6.2.3 User interface

For the project purpose, the scores that were to be converted into guitar tablatures were manually translated into Python data structures. This is an unacceptable step for end-user. The translation could be carried out automatically from some generic music representation (like musicXML format for example) to required Python representation. At the moment the best individual is printed out to the screen, there is obviously a room for improving this from usability point of view.

6.3 Personal learning outcomes

The project was an excellent opportunity to learn new technologies and ideas.

Learning Python programming language was a very rewarding experience. I was exposed to a thriving community of enthusiast of the programming language that are

behind inexhaustible resources on any aspect of the subject. Coming from a strongly typed language background, it was very refreshing to see how easier problems can be approached in a dynamically typed language like Python resulting in less verbose and neater code. At the same time I felt my Java background was a hindrance in that I tended to code algorithms as I would in Java which made my code look less 'pythonesque' (a term used by Python programmers to describe a code that leverages Python specific idioms and features).

Learning principles of Genetic Algorithms and using these principles to approach a practical problem was very challenging, as there were so many new elements that had to be brought together in order to do that.

Finally, undertaking the project that mixed such diverse disciplines as computer science, biology and music, even if the end result is not 'production ready', was certainly very gratifying.

7 References

Aleksander Radisavljevic. (2004). Path Difference Learning for Guitar Fingering Problem. Retrieved August 26, 2012, from http://hdl.handle.net/2027/spo.bbp2372.2004.141

D. R. Tuohy. (2005). A GENETIC ALGORITHM FOR THE AUTOMATIC GENERATION OF PLAYABLE GUITAR TABLATURE. Retrieved August 26, 2012, from http://hdl.handle.net/2027/spo.bbp2372.2005.013

Fortin, F.-A., Rainville, F.-M. D., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, *13*, 2171−2175.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Mitchell, M. (1998). An Introduction to Genetic Algorithms. Retrieved from

http://mitpress.mit.edu/catalog/item/default.asp?tid=5974&ttype=2

Python Programming Language – Official Website. (n.d.). Retrieved August 27, 2012, from http://python.org/

Rutherford, N. (2009). FINGAR, a Genetic Algorithm Approach to Producing Playable Guitar Tablature with Fingering Instructions.

Sayegh, S. I. (1989). Fingering for String Instruments with the Optimum Path Paradigm. *Computer Music Journal*, *13*(3), 76. doi:10.2307/3680014

Sivanandam, S. N., & Deepa, S. N. (2010). *Introduction to Genetic Algorithms* (Softcover reprint of hardcover 1st ed. 2008.). Springer.

TabWiki. (n.d.). Retrieved August 26, 2012, from http://www.tabwiki.com/index.php/Main_Page

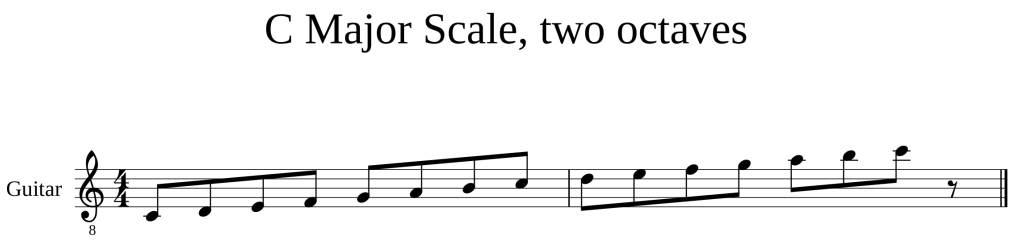
Tuohy, D. R., & Potter, W. D. (2006). Generating Guitar Tablature with LHF Notation Via DGA and ANN. *Lecture notes in computer science.*, (4031), 244–253.

8 Appendices

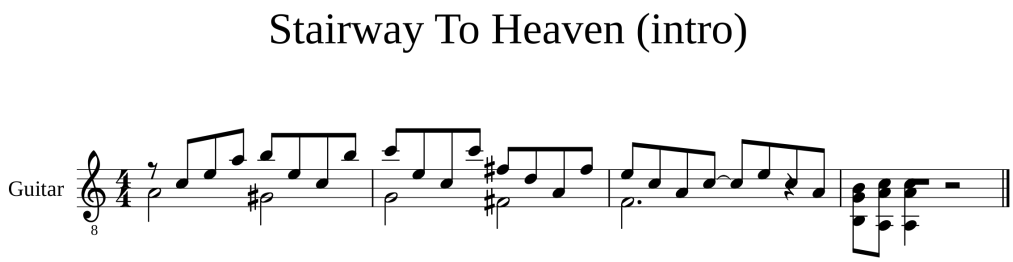
8.1 Input Scores

All scores were created using MuseScore - an open source music notation software (http://musescore.org).

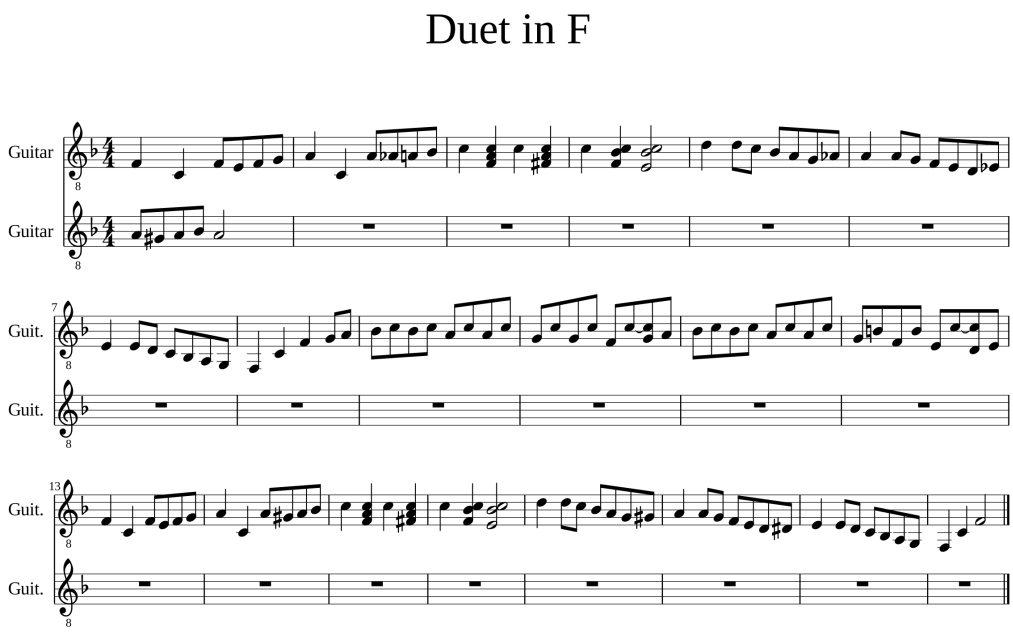
8.1.1 C major scale



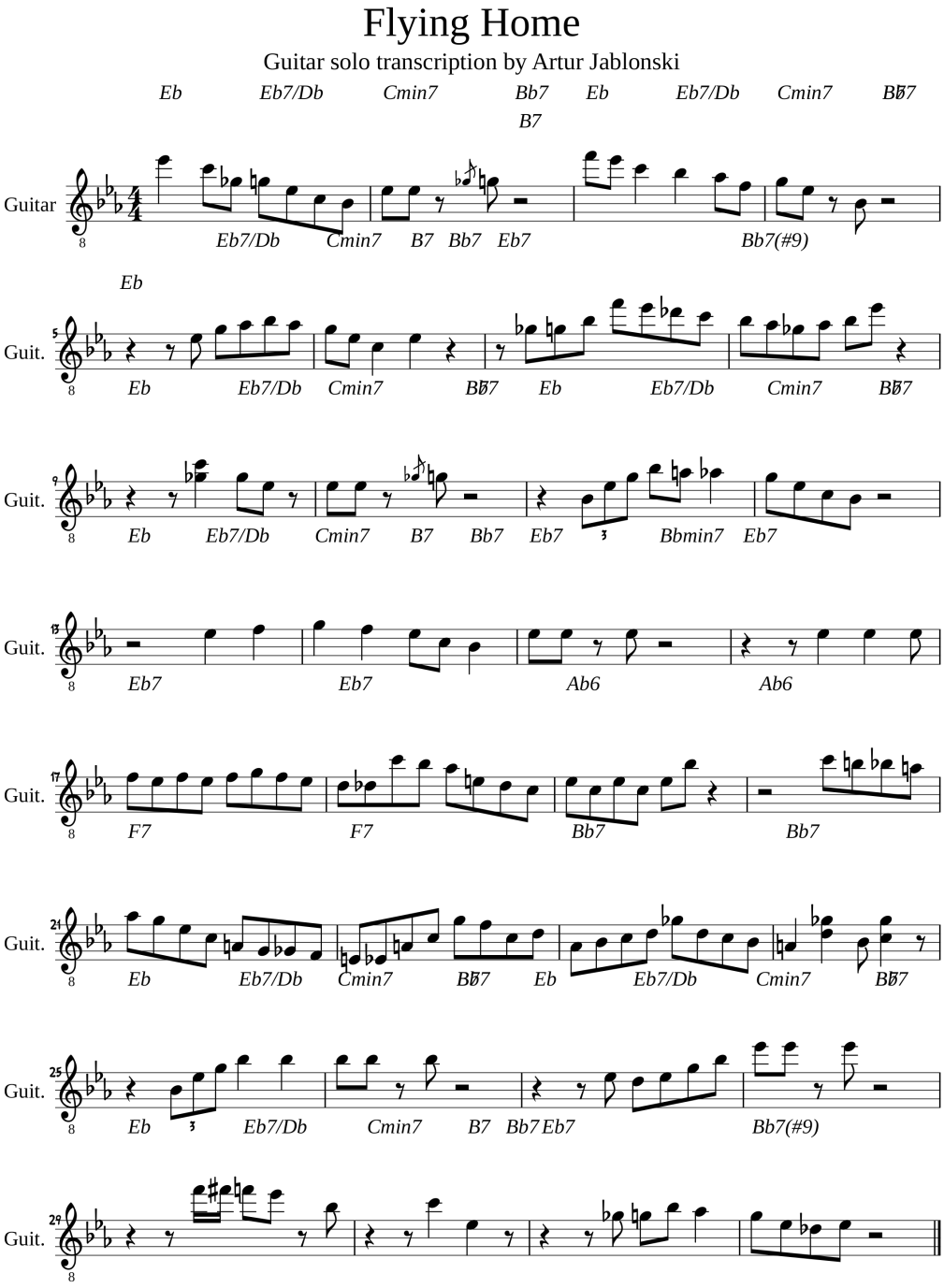
8.1.2 Stairway To Heaven



8.1.3 Duet in F



8.1.4 Flying Home



8.2 Output tablatures

8.2.1 C Major scale

```
Individual 1:
Fitness  (3.7333333333333334, 2.8181818181818183)
E---------------------------------1--3--5--7--8-
B--------------------0--------5---------------
G---------------0--2-----5--7------------------
D------0--2-------------------------------------
A---3--------8-----------------------------------
E-------------------------------------------------

Individual 2:
Fitness  (3.8666666666666667, 2.4166666666666665)
E-----------------------------------3--5--7--8-
B--------------------0--1--3--5--6-------------
G---------------0-------------------------------
D---------2--3-----7---------------------------
A---3--5----------------------------------------
E-------------------------------------------------

Individual 3:
Fitness  (4.2000000000000002, 2.2307692307692308)
E-----------------------------------3--5--7--8-
B--------------------0--1--3--5--6-------------
G-------------------------------------------------
D---------2--3--5--7---------------------------
A---3--5----------------------------------------
E-------------------------------------------------

Individual 4:
Fitness  (4.333333333333333, 2.0833333333333335)
E-----------------------------------3--5--7--8-
B--------------------0----------6-------------
G------------------2-----5--7--9---------------
D------0--2--3--5-------------------------------
A---3-------------------------------------------
E-------------------------------------------------

Individual 5:
Fitness  (4.666666666666667, 2.0)
E-----------------------------0-----3--5--7--8-
B---------------------------------6-------------
G------------------2--4--5--7------------------
D------0-----3--5-------------------------------
A---------7-------------------------------------
E---8---------------------------------------------
```

```
Individual 6:
Fitness  (5.0, 1.8333333333333333)
E---------------------------------------5--7--8-
B-------------------------------5--6--8----------
G---------------0--2--4--5--7------------------
D------0-----3---------------------------------
A---------7-----------------------------------
E---8-----------------------------------------

Individual 7:
Fitness  (6.666666666666667, 1.8181818181818181)
E-------------------------------0--------------8-
B---------------------------------6--8--10-12---
G---------------0-----------7--------------------
D------0-----------7--9--10----------------------
A---------7--8---------------------------------
E---8-----------------------------------------
Individual 8:
Fitness  (7.333333333333333, 1.6923076923076923)
E-----------------------------------------------
B-------------------------------5--6--8--10-12-13
G---------------0-----------7--------------------
D------------------7--9--10----------------------
A---3--5--7--8---------------------------------
E-----------------------------------------------
```

8.2.2 Stairway To Heaven

```
Individual 1:
Fitness  (4.083333333333333, 2.7619047619047619)
E---------0--5--7--------7--8--------8--2--------2--0-------------
B------------------5-----------5--1--------3-----------1----------
G---2--5--------------5----------------------2--------------5--9-
D---------------6-----------5-----------4-----------------7-------
A---------------------------------------------------------------
E-------------------------------------------------13------------

E-----------------
B---------0--1--1-
G---------0--2--2-
D---10------------
A------12-2--0--0-
E-----------------

Individual 2:
Fitness  (4.25, 2.6666666666666665)
E---------0--5--7--------7--8--------8--2--------2--0-------------
B------1-----------5-----------5--1--------3-----------1--------5-
G---------------------5----------------------2--------------5----
D---7-----------6-----------5-----------4-----------------7-------
A---------------------------------------------------------------
E-------------------------------------------------13------------

E-----------------
B------------1--1-
G---------4--2--2-
D---10-7--5-------
A------------0--0-
E---------7-------

Individual 3:
Fitness  (5.3888888888888893, 2.5238095238095237)
E---------0--5--7--------7--8----------------------0-----------0-
B------------------5-----------5--------7--------7-----------1----
G---2--5--------------5--------------17----7-----------5----------
D---------------6-----------5-----10----4-----7-----------7-------
A---------------------------------------------------------------
E-------------------------------------------------13------------

E-----------------
B---------0--1----
G---5--------2--5-
D------7--------7-
A---------10-0----
E---------7-----5-
```

```
Individual 4:
Fitness  (5.666666666666667, 2.4782608695652173)
E------------5--7-----------8--------8-----------2--0------------
B---------5--------5-----12----5----------------------1--------5-
G---2--5--------------------0-----5-----11-------------------5----
D---------------6-----10-------------------12-7-----------7-------
A---------------------------------------------------------------
E-----------------------------------14----------13------------

E-----------------
B------------1--1-
G---------4--2--2-
D---10-7--5-------
A------------0----
E---------7-----5-

Individual 5:
Fitness  (5.7777777777777777, 2.4090909090909092)
E---------0--5--7--------7--8----------------------0----------0-
B------------------5----------5--------7--------7----------1----
G---2--5--------------5--------------17----7----------5----------
D---------------6----------5-----10----4-----7----------7-------
A---------------------------------------------------------------
E-------------------------------------------------13------------

E-----------------
B---------0-------
G---5--------5--5-
D------7-----7--7-
A---------10------
E---------7--5--5-

Individual 6:
Fitness  (6.1944444444444446, 2.3809523809523809)
E------------5--7--0--------------------------------0----------0-
B---------5--------------12-13----------7--------7----------1----
G------5----------------------9-----17----7----------5----------
D---7----------6-----10----------10----4-----7----------7-------
A--------------------------10---------------------------------
E-------------------------------------------------13------------

E-----------------
B---------0--1----
G---5--------2--5-
D------7--------7-
A---------10-0----
E---------7-----5-
```

8.2.3 Duet in F

```
Individual 1:
Fitness  (5.6521739130434785, 5.0272727272727273)
E-------------------------------------------------------------------
B-----------------------------------1-----1--1-----1--1--3--3-
G-----------------0--2----------2--3-----5-----2--5--3--3-------
D---3-----3---------------------------7-----4-----3--2-------
A------------7----------3--12-11----------8---------------------
E------8--------13------------------------------------------------

E-------------------------------------------------------------------
B-------------------------------------------------------------------
G---------2-----1-----2-------------------------------------------
D---10-------5-----7-----3--------0-------------------------------
A------13-----------------8--------6--7--7-----3--1-----------3-
E-----------------------------12-------------10-------5--3--1----

E-------------------------------------------------------------------
B--------------------------1-----------------1-----1-------------
G------0--2-----5-------------2----------------------0--2--3----
D---3--------8-----8-----------------5--------------------------10
A---------------------15-12-------15----15-10----8---------------
E-------------------------------------------------------20---------

E-------------------------------------------------------------------
B-----------------------------0--------1-------------------------
G------5--2--5-----------------------------------------------2-
D---------------7-----------------2--10-0--2-----------------5----
A---13----------------------8----------------8--3--8-----8-------
E-----------------20-15-19---------------------------12---------

E-------------------------------------------------------------------
B------------------1--1--1--1--------1-----3---------------------
G--------------------2-----2-----5--3-----------3--2-------------
D------7--6-----------3-----4--10-8--2-----------------5--6--7----
A---3-----------13----------------8-----------15----------------12
E------------17-------------------------22-----------------------

E-----------------------------------------------
B-----------------------------------------------
G-----------------------------------------------
D---3-----2--0-----------0---------------------
A--------------------------3--1--0-----------8-
E------13-------11-12-12-------------3--1--8----

Individual 2:
Fitness  (5.7971014492753623, 4.6759259259259256)
E-------------------------------------------------------------------
B-----------------------------------------1--1-----1-----1--3--3-
G--------------------2-----2-----2--3-----2-----5-----5--3-------
D---------3--------5-----------6--------10-3-----7-----8--2-------
A---8--------------------3----------------------9-----8----------
E------8-----12-13------------------------------------------------
```

```
E-------------------------------------------------------------
B---1---------------------------------------------------------
G---------2--0--------2---------------------------------------
D-------------------------3--3--2--0-----------0-------------
A------13-----------------------------6--7--7-----3--1-------3-
E---------------16-17-----------------------------------5--3--1----

E-------------------------------------------------------------
B---------------1-----1-----------1-----1-----------1---------
G---------2--------3-----2-----------------------------0--2-------
D---3--5-----8-----------------------------5--10-----------------
A---------------------------15-------10----------8-----------13-15
E-------------------------------17---------------------20---------

E-------------------------------------------------------------
B------1-----1-------------------------------------------------
G------------------5--0--4--------------5--------------------0----
D---------------7--------------9-----10-0-----------------------
A---------12----------------------7--------7--------8--7--------12
E---18----------------------13----------------13-8--------13------

E-------------------------------------------------------------
B---------------------------1-----1-----3--3---------------------
G------2-----------5--5--5--2-----3--5----------------0-----2----
D---------------------7-----4--10-3--8--------------------6-------
A---3-----11-12-13----8--------------7--------15----------------12
E-----------------------------------------------18-17------------

E-----------------------------------------------
B-----------------------------------------------
G-----------------------------------------------
D------3-----0-----------0-------------------3-
A---------------6--7--7-----3-----0-------------
E---13----12-------------------6-----3--1--8----

Individual 3:
Fitness  (5.86231884057971, 4.5357142857142856)
E-------------------------------------------------------------
B---------------------------------------1--1-----1--1--1--1--3----
G------------------0--2--------------------2--5--2-----3--3-----7-
D---3-----3-----3-------------6--7--8-----3-----4-----3--2-------
A------------7-----------3--12-------------------------------------
E------8-------------------------------------------------------

E-------------------------------------------------------------
B-------------------------------------------------------------
G---5-------------2--2-----------------------------------------
D-------------------------3-----------------2-----------------------
A------13----10----------------7--5--6--7-----5--------0----------
E---------17----16----------13-------------------8--6-----3--1--8-
```

```
E-------------------------------------------------------------
B---------------------------------1-----1-----------1---------
G---------------5-------------2-----0-----------------0--2-----
D------5--7-----------10-------------------5--10---------------
A------------------13----12-15--------------------8-----------13-15
E---13-------18-----------------------------------------20---------

E-------------------------------------------------------------
B------1-----1-------------------------------------------------
G-----------------5--0--4-----------------------------------0--2-
D---------------7-------------9-----10-10-------------2----------
A---------12-------------------------------------3--8-----8-------
E---18----------------------13----12----10-12-13-----------------

E-------------------------------------------------------------
B-----------------1--1-----------1--1--------1-----------------
G------2--------------2-----------3--3--7--7-----3--2--------2--2-
D---------------8-----3--10-10-10-3--2--------------------6-------
A------------12-------------12---------------------------------
E---8-----16----------------14-------------------------15---------

E-----------------------------------------------
B-----------------------------------------------
G-----------------------------------------------
D---3--3-----0--------2----------------------3-
A---------7-----6--------5--3--1---------------
E------------------12-------------5--3--1--8----

Individual 4:
Fitness  (5.8913043478260869, 4.3090909090909095)
E-------------------------------------------------------------
B---------------------------------1--1-----1--1-----1--3--3-
G-----------------0--2--------------------2--5--2-----5--3-------
D---3-----3--2--3--------------6--7--8-----3-----4-----8--2-------
A------------------------3--12-------------------------8----------
E------8-------------------------------------------------------

E-------------------------------------------------------------
B---1---------------------------------------------------------
G---------2--0--------2---------------------------------------
D------------------------3--3--2--------------0---------------
A------13-------------------------5--6--7--7-----3--1----------3-
E---------------16-17-----------------------------------5--3--1----

E-------------------------------------------------------------
B---------------------------------1-----------------------------
G---------------5-------------2-----0--5--0--5--------5----------
D------5--7-----------10-7-----------------------------5-----8----
A------------13----13-------15----------------------15----12----15
E---13-----------------------------------------13--------------
```

```
E-------------------------------------------------------------
B-----------------------------0-----1--1-----------------------
G------5--2-----------0-----------------------------------------2-
D---8--------------------9--------------0-----------------------
A------------15-12-15----------------------7--8-----8------------
E--------------------------13----12-------------8-----12-13-15---

E-------------------------------------------------------------
B--------------------------1-----1--1--3-----------------------
G---------1--2--3----------2--5--3--3--------5--------------2----
D---------------------10-10-4-----3--2----------8--7----------7-
A---3--12----------15-12-------------------17-------------11------
E---------------------13-------------------------------15---------

E-----------------------------------------------
B-----------------------------------------------
G-----------------------------------------------
D---------2--0-------------------------------3-
A---8--8--------6--7--7-------------------------
E------------------------10-8--6--5--3--1--8----

Individual 5:
Fitness  (5.9420289855072461, 4.3008849557522124)
E-------------------------------------------------------------
B---------------------------------------1-----1--1-----1--1--3--3-
G------------------0--2-----------2--3-----5-----2--5--3--3-------
D---------------3--------------------------7-----4-----3--2-------
A------------------------3--12-11----------8---------------------
E---13-8--13-12-----------------------------------------------

E-------------------------------------------------------------
B-------------------------------------------------------------
G---------2-----1-----2---------------------------------------
D---10-------5-----7-----3--------0-----------------------------
A------13-------------------8--------6--7--7-----3--1-----------3-
E-----------------------------12-------------10-------5--3--1----

E-------------------------------------------------------------
B--------------------------1-----------------------------------
G------0--2-----5--------------2--------------5--------------3----
D---3--------8-----8-----------------5--------------10-10---------
A--------------------15-12-------15----15-10----8-----10-------15
E-------------------------------------------------------17------

E-------------------------------------------------------------
B-----------------------------0--------1------------------------
G------------5-----5-----------------5---------------------------
D------10-7-----------5--9-------------0-----------3--2--3-------
A---13----------12-------------------------7--8--3--------------12
E--------------------------13----12-------------------------15---
```

```
E-------------------------------------------------------------
B---------------------------1-----1--------3-----------------
G------2-----2--3-----------2--5--3--5--7--------------0--1--2----
D---------------------10-10-4-----3--8--------------7-----------7-
A---3-----11-------15-12-------------7-----------13--------------
E--------------------13----------------------20-----------------

E-----------------------------------------------
B-----------------------------------------------
G-----------------------------------------------
D---------2-----1--2--------------------------3-
A---8--------5--------7--5--3--1----------------
E------13-------------------------5--3--1--8----

Individual 6:
Fitness  (5.9927536231884062, 4.1559633027522933)
E-------------------------------------------------------------------
B-----------------------------------------1--1-----1-------------
G--------------------2-----2-----2--3-----2-----5-----5--------7-
D---------3--------5-----------6--------10-3-----7-----8--10-12---
A---8--------------------3----------------------9-----8--13------
E------8-----12-13-----------------------------------------12------

E-------------------------------------------------------------
B-------------------------------------------------------------
G---5-----2--------2--2---------------------------------------
D------------5--6-----------------0-----------------------------
A------13----------------8-----7-----6--7--7--5--------0----------
E--------------------------13-------------------8--6-----3--1--8-

E-------------------------------------------------------------
B--------------------------1-----1-----1-----1--------1--------1-
G-----------------3----------2----------0----------0----------
D------------------------7-------------------------10----7--8----
A---8--10-12-13-15--------------------10--------------------------
E---------------------20-------------------------13--------------

E-------------------------------------------------------------
B------1----------------------0--------1-----------------------
G-----------------------------------5----------------------0----
D---8--------10-7--------9--------2-----0-----------------------
A---------12----------------8--------------7--8-----8--7--------12
E------------------20-15------------------------8--------13------

E-------------------------------------------------------------
B--------------------------------1--1--------1-----------------
G------2-----------5--5--5--------3--3--7--7-----3--2--------2--2-
D---------------------7-----10-10-3--2--------------------6-------
A---3-----11-12-13----8-----12-----------------------------------
E--------------------------14-------------------------15---------
```

```
E-------------------------------------------
B-------------------------------------------
G-------------------------------------------
D---3--3-----0-----------0-------------------3-
A---------7-----6--7--7-----3--1--0-------------
E-----------------------------------3--1--8----
```

```
Individual 7:
Fitness  (6.2608695652173916, 3.9727272727272727)
E-----------------------------------------------------------------
B-----------------------------------------------1--1--------1-------
G------------------0--2-----------2--3-----5-----2--5--5--3--7--7-
D---3--------------------------6--------10-7-----4-----8--2-------
A---------8-----------------12-------------8-----------8----------
E------8-----12-13-------8---------------------------------------
```

```
E-----------------------------------------------------------------
B-----------------------------------------------------------------
G---5--------0-----2--2-------------------------------------------
D------8-----------------3-----2--------2--2----------------------
A---------12----------------8-----5--6-----------------0--------3-
E---------------16----------------------------10-8--6-----3--1----
```

```
E-----------------------------------------------------------------
B---------------------1-------------------------------1----------
G------0--2-----5--------2--------5-----------5--------0--2--3--5-
D---3--------8--------------10-------5--------------10------------
A-----------------------------12-------15-------8----------------
E------------------18----------------------15--------------------
```

```
E-----------------------------------------------------------------
B------1-------------------------------------------------------
G---3--------------------4-----------5----------------------0--2-
D------------10----10-5-----------------0-----3-----3------------
A---------12----12-------------14-7-----15-7-----3---------------
E---------------------------13-------------------------12-13------
```

```
E-----------------------------------------------------------------
B---------------------1-------------------------------------------
G------2-----2-----5--2--------------5--7--------3---------------
D---------------8-----3--10-10----10-8-----12-------7------------
A---3-----11----------------12-15-13-7--------15-------10-11-12-12
E---------------------------14----13-----------------------------
```

```
E---------------------------------------------
B---------------------------------------------
G---------------------------------------------
D---3--------0-----2--2--0---------------------
A---------------6-----------3--1--------------8-
E------13-12----------------------5--3--1--8----
```

8.2.4 Flying Home

```
Individual 1:
Fitness  (10.19620253164557, 5.7712418300653594)
E---11-------------------------3--------8-------------------------
B-----------------------4--4-----18-----------------------------
G------17-11-12----------------------20----15-------12----------12
D---------------13-10-------------------------18-15----13----13---
A-------------------------------------------------------------------
E---------------------18-----------------------------------18------

E---------4----------------------13----------6--4--2--4--6--11---
B---9--------------------------------16-------------------------13
G------15----12-8-----8-----12----------18-17-------------------11
D------------------10----16----20---------------------------------
A-------------------------------------------------------------------
E-------------------------------------------------------------------

E---------------3--------------5---------------------------------
B------4-----4-----------8--11----------------------------6-------
G---------------------------------13-12-8--5----------------8----
D---17----13-------8-----------------------------13--------------
A---------------------18----------------------13----20-22-------15
E-------------------------------------------------------------------

E---------------------------------------3--------------8--------0-
B------4--4-----------4--6-----6-----------6--------------11-9----
G---------------8--8--------------8-----------8-------------------
D---------------------------13--------------------12--------------
A---13-------18----------------------20-------------16------------
E-------------------------------------------------------------------

E------------------------6-----7--6-----------------------------
B---------4-----------4-----13-------10-9------------------------
G------5-----------5-------------------------------------------
D------------10-13-------------------------------10----5----------
A---17-------------------------------------22-------------9--8--7-
E-----------------------------------------------23----17----------

E---------------1-------------------------------------------------
B---------1--8--------3-----------3-----------------7-----------4-
G-----------------------------------------5-----2--7--3--11-3----
D---1--------------10----6-----------16-12-----------------------
A-------------------------------15-------------------------15------
E------17-------------------18----------------18-----------------

E---3--6-----6-----6--------------------11-11----14-13------------
B---------------11----4-----------11-16----------------------13-4-
G---------15-------------7--------------------22-------20-15------
D--------------------------13-----------------------------------
A-------------------------------22-----------------------------
E-------------------------------------------------------------------
```

```
E------3--------3----------
B---7-----11-9-----4-------
G---------------------6----
D--------------------------
A----------------------18--
E--------------------------

Individual 2:
Fitness  (10.49367088607595, 5.3071895424836599)
E------8--2--------------------3-----11-8--6----------------------
B--------------------------4-----18-------------------------------
G---20-------12-8--------8---------------------------------8----
D-------------------------------------------------17----------17
A--------------------13----------------------23-20----18---------
E------------------20-----------------------------------18------

E------------------------2--3--------------------4---------------
B------11-------4-----4-----------18----------11----7-----------13
G------------12----5-----------------20----------------13-15-20---
D---18----18-------------------20-------23-22-------------------16
A------------------------------------------------------------------
E------------------------------------------------------------------

E--------------------------6----------------------1-------------
B---------------8--------------10-9--------------4-----8--------1-
G---------8--8--------8--12----------12-8-----3-------------------
D---17-13-----------------------------------------------------13---
A------------------13----------------------15-------------20------
E------------------------------------------------------------------

E------------------------------1-----------1-----------8-----4----
B--------------------------------4--6--8-----4-----2-----------5-
G------------8--8--------10-8-----------------------------15------
D---8-----13-------13-13-----------------------------------------
A------18--------------------------------------------------------
E-------------------------------------------------22--------------

E---------------------------------6------------------------------
B---3-----------4--------11-13-12-------9------------------------
G------------5-----------------------------12----------0----------
D------10-13----------13----------------------13-10-------4-------
A------------------15----------------24--------------------------
E-------------------------------------------------17-------13-12

E-----------------------------------2--------------2-------------
B-----------------------------1--3-----------------3-------------
G---------5--12-------7-----------------7--5-----2--------11-3--8-
D------7--------15-------6--------------------8--------8--10------
A------------------------------------------------------------------
E---11-------------20-------18-----------------------------------
```

```
E--------------6----------------6-----------13-------11---------
B---8--11-11-11------------------------16-16----19-------11------
G------------------15----------------20-------------22-------17-8-
D--------------------13-12-13-17---------------------------------
A---------------------------------------------------------------
E---------------------------------------------------------------

E-------------------------
B---7--8--11-9--8--4-----4-
G-------------------------
D-------------------------
A-------------------------
E--------------------21---
```

Individual 3:
Fitness (10.670886075949367, 4.9155844155844157)

```
E---11-------------------------------11---------------------------
B------13----8--4--1-----4-----8--18-----------------------------
G---------11---------------8--------------------10-12----------12
D--------------------8----------------22-20----------13----13---
A-----------------------------------------------23----------------
E---------------------------------------------------------18------

E---------4----------------------13----------6--4--2--4--6--11---
B---9-------------------------------------------------------------13
G------15----12-8-----8-----12-------20-18-17-------------------11
D------------------10----16----20---------------------------------
A---------------------------------------------------------------
E---------------------------------------------------------------

E---------------3--------------5--4-----------------------------
B---8-----4-----------4-----11-------8--4--1--------6--8----------
G-----------------------------------------------8--------10------
D------13----13----------17-------------------8-----------------10
A-------------------------------------------------------------18---
E------------------18-----------------------------------------------

E-----------------------------------------1----------8----------
B-------------------------------6--4-----------4--3--2-----11----5-
G------------------------10-8--------10---------------------------
D---8--------------13-----------------------------------------18---
A------18-18-18-18----------------------22-----------------------
E--------------------23-----------------------------------------

E---------------------------------------------------------------
B---3--1-----------------11-------------------------------------
G------------5--------8-----17-16-15-14-13----------2-----------
D---------13-------10----------------------17----------5--4-------
A---------------18-----------------------------18-------------8--7-
E-----------------------------------------------20---------------
```

```
E-----------------------------------2-------------2------------
B------------8-------------------------------------3-----7-----4-
G------2--5---------------------------------5-------------5-------
D---------------15----------------12----12-------7---------------
A------------------15-17-------15-------------13-------13----13---
E---11------------------16-18-------------------------------------

E---------6-----------------------------------13-----------------
B---8--11-------------4--3-----8-----------------19-18-16---------
G---------------15-15-------8-----15-20-20-20-------------15------
D------------20-------------------------------------------------22---
A---------------------------------------------------------------------
E---------------------------------------------------------------23

E---------6--4--3----------
B------8-----------4-----4-
G---11----------------6----
D--------------------------
A--------------------------
E--------------------------

Individual 4:
Fitness  (10.772151898734178, 4.7467532467532472)
E---11-------------------------------11--------------------------
B------13----8--4--1-----4-----8--18-----------------------------
G---------11----------------8--------------------10-12----------12
D---------------------8-----------------22-20----------13----13---
A-------------------------------------------23-----------------
E-----------------------------------------------------18------

E---------4----------------------13----------6--4--2--4--6--11---
B---9-------------------------------------------------------------13
G------15----12-8-----8-----12-------20-18-17------------------11
D------------------10----16----20-----------------------------
A---------------------------------------------------------------
E---------------------------------------------------------------

E---------------3-------------5--4-----------------------------
B---8-----4-----------4-----11-------8--4--1--------6--8----------
G-------------------------------------------8--------10------
D------13----13----------17-------------------8-----------------10
A-------------------------------------------------------18---
E------------------18-----------------------------------------

E---------------------------------------1----------8----------
B----------------------------6--4----------4--3--2-----11----5-
G----------------------10-8--------10-------------------------
D---8-------------13-----------------------------------18---
A------18-18-18-18---------------------22----------------------
E--------------------23-----------------------------------------
```

```
E-------------------------------------------------------------
B---3--1-----------------11------------------------------------
G------------5--------8-----17-16-15-14-13----------2----------
D---------13-------10----------------------17----------5--4-------
A---------------18----------------------------18-------------8--7-
E-----------------------------------------------20--------------

E-----------------------------------2----------------------------
B------------8-----------------------------------------7-----7-------
G------2--5---------------------------------5--------7--3--------8-
D---------------15----------------12----12----------------10------
A------------------15-17-------15-------------13-12----------13---
E---11-------------------16-18-------------------------------------

E---------6-----------------------------------13------------------
B---------------------4--3-----8-----------------19-18-16---------
G---12-15----15-15-15-------8-----15-20-20-20-------------15------
D-------------------------------------------------------------22---
A-------------------------------------------------------------
E-------------------------------------------------------------23

E---------6--4--3----------
B------8-----------4-----4-
G---11----------------6----
D--------------------------
A--------------------------
E--------------------------
```

8.3 Source code

8.3.1 gentab.py

```
# Author: Artur Jablonski
# Date: Sept 2012
#
# This software is submitted in partial fulfillment of the requirements for the
# BSc (Honours) Information Systems and Information Technology (DT249)
# to the School of Computing, Faculty of Science, Dublin Institute of Technology.

'''
gentab.py
This is the main module and entry point of the algoirhm
It uses modules from deap, core and input to implement
genetic algoirhtm that converts music scores into guitar tablatures
'''

from deap import base, creator, tools
import core
import random

import time
import matplotlib.pyplot

import input as scoreInput

#input module contains manuall translated scores


FITNESS_WEIGHTS=(-1.0,-1.0)

#INPUT_SCORE = scoreInput.cMajorScale
INPUT_SCORE = scoreInput.stairway2heaven
#INPUT_SCORE = scoreInput.duetInF
#INPUT_SCORE = scoreInput.flyingHome


creator.create("FitnessMax", base.Fitness, weights=FITNESS_WEIGHTS)
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()

# Structure initializers
toolbox.register("individual", core.initTablature, creator.Individual, INPUT_SCORE)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Operator registering
toolbox.register("evaluate", core.tabMultiFitness)
toolbox.register("mate", tools.cxTwoPoints)
toolbox.register("mutate", core.tabMutate, indpb=0.05)
toolbox.register("select", tools.selSPEA2)



def main():

    random.seed(64)

    #start the clock
    t0 = time.clock()

    pop = toolbox.population(n=300)
    hof = tools.ParetoFront()

    stats = tools.Statistics(key=lambda ind: ind.fitness.values)
```

```
stats.register("avg", tools.mean)
stats.register("std", tools.std)
stats.register("min", min)
stats.register("max", max)


CXPB, MUTPB, NGEN = 0.5, 0.2, 50

# Evaluate the entire population
fitnesses = map(toolbox.evaluate, pop)
for ind, fit in zip(pop, fitnesses):
    ind.fitness.values = fit

hof.update(pop)
stats.update(pop)


column_names = ["gen", "evals"]
column_names += stats.functions.keys()
logger = tools.EvolutionLogger(column_names)
logger.logHeader()
logger.logGeneration(evals=len(pop), gen=0, stats=stats)



# Begin the evolution
for g in range(1, NGEN + 1):


    # Select the next generation individuals
    offsprings = toolbox.select(pop, len(pop))
    # Clone the selected individuals
    offsprings = map(toolbox.clone, offsprings)

    # Apply crossover and mutation on the offsprings
    for child1, child2 in zip(offsprings[::2], offsprings[1::2]):
        if random.random() < CXPB:
            toolbox.mate(child1, child2)
            del child1.fitness.values
            del child2.fitness.values

    for mutant in offsprings:
        if random.random() < MUTPB:
            toolbox.mutate(mutant)
            del mutant.fitness.values

    # Evaluate the individuals with an invalid fitness
    invalid_ind = [ind for ind in offsprings if not ind.fitness.valid]
    fitnesses = map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit



    hof.update(offsprings)

    # The population is entirely replaced by the offsprings
    pop[:] = offsprings

    stats.update(pop)


    logger.logGeneration(evals=len(invalid_ind), gen=g, stats=stats)

#stop the clock
t1 = time.clock()

print "Running time: ", t1 - t0, " seconds"
```

```
    for pareto, indx in zip(hof, range(1, len(hof) + 2)):
        print "Individual %d:" % indx
        print "Fitness ", pareto.fitness.values
        core.printTablature(pareto, widthMax=21)

    #plot graph
    statsMinimum = stats.data.get('min')[0]
    fretPositionMinimum = [x[0] for x in statsMinimum ]
    fretDistanceMinimum = [x[1] for x in statsMinimum ]

    matplotlib.pyplot.scatter(range(0,len(fretPositionMinimum)), fretPositionMinimum,
c='b', marker='p', label="average fret position")
    matplotlib.pyplot.scatter(range(0,len(fretDistanceMinimum)), fretDistanceMinimum,
c='r', marker='p', label="average fret distance")
    matplotlib.pyplot.xlabel("Generation")
    matplotlib.pyplot.ylabel("Minimum fitness value")
    matplotlib.pyplot.xlim(xmin=0)
    matplotlib.pyplot.legend()
    matplotlib.pyplot.show()


if __name__ == "__main__":
    main()
```

8.3.2 core.py

```
# Author: Artur Jablonski
# Date: Sept 2012
#
# This software is submitted in partial fulfillment of the requirements for the
# BSc (Honours) Information Systems and Information Technology (DT249)
# to the School of Computing, Faculty of Science, Dublin Institute of Technology.
'''
core.py

This module contains a set of functions that supports the gentab.py module
in implementing its logic.

It implements, amongs other things, mutation operator and initialization of random
tablature code
'''

import sys
import itertools
from random import choice,random

#tablatrue is represented as a list of lists of tuples
#the top level lists contain following notes to play
#the list of tuples contain all simultaniously played notes
#tuples have form (a,b) where a is fret number and b is string number
#string are numberered from 0 to 5 where 0 is the low and 5 is the high E

tablature = [[(1,5)],
             [(2,4)],
             [(6,0)],
             [(7,5), (7,4), (7,3)]
            ]


#notes are represented as
#a list of lists of tuples
#each tuple is of a form (a,b,c) where
#a is the note pitch
#b indicates accidentals ('#' for sharp, 'b' for flat, 'n' for natural)
#c denotes octave number
#the lowest note on standard tuned guitar is E3

cMajorScale = [[('C','n', 4)],
               [('D','n', 4)],
               [('E','n', 4)],

              ]

def getValidChordsForFingerings(chordFingerings):
    '''
    This function returns all valid chord fingerings for a given instance
    of a chord.
    A valid chord is considered to be a chord not spanning more than 4 frets

    '''
    result = list()
    fingeringsList = list()
    for note in chordFingerings:
        fingeringsList.append(getAllFingerings(note))

        #generate cartesian product from the fingerings
        cartesianProduct =  itertools.product(*fingeringsList)

        validFingerings = list()
        for chord in cartesianProduct:
```

```
            strings = list()
            invalid = False
            for note in chord:
                if (note[1] in strings):

                    invalid = True
                    break
                else:
                    strings.append(note[1])
            if getChordFretSpan(chord) > 4:
                invalid = True

            if invalid == False:
                validFingerings.append(chord)
        result = validFingerings

    return result


def initTablature(container, classicalScore):
    '''
    initializes tablature from a classical notated representation
    to a tablature with random fingerings
    '''
    result = container()

    for timeslot in classicalScore:
        tabTimeslot = list()

        #if there is one note in the slot then choose fingering randomly
        if len(timeslot) == 1:
            fingerings = getAllFingerings(timeslot[0])
            #choose one fingering randomly
            tabTimeslot.append(choice(fingerings))
        else: #otherise things are tricky, as we cannot choose two fingerings on the
same string...
            #You can't play two notes on the same string!
            #first check if it's even possible to have fingerings on different strings
            validFingerings = getValidChordsForFingerings(timeslot)


            tabTimeslot = list(choice(validFingerings))

        result.append(tabTimeslot)

    return result

def getAllFingerings(noteTuple):
    '''
    takes a note tuple and returns tuple of possible tablature fingerings
    for a standard tuned 24 fret guitar
    '''

    #assume E3 is number 0, then get a distance of semitones from E3
    if not noteTuple:
        return list()

    distances = {'C': -4,
                 'D': -2,
                 'E': 0,
                 'F': 1,
                 'G': 3,
                 'A': 5,
                 'B': 7
                }
    #first get distance from E in the same octave:
    distance = distances[noteTuple[0]]
    #consider accidentals
```

```
    if noteTuple[1] == '#':
        distance += 1
    if noteTuple[1] == 'b':
        distance -= 1

    #now add the octave difference
    distance += 12 * ( noteTuple[2] - 3)
    #print distance
    #having the total distance in semitones from E3, it's easy to calculate
    #distances from each string on guitar
    substractFactor = (0,5,5,5,4,5)
    fingerings = list()
    for (stringNo, sF) in zip(range(0,6), substractFactor):
        #print distance
        #print stringNo
        #print sF
        distance -= sF
        if distance < 0:
            break
        if distance <= 24:
            fingerings.append((distance,stringNo))


    return fingerings


def getChordFretSpan(chord):
    '''
    for a given chord returns an interger that represents how many
    frets does the chord fingering span.
    '''
    if len(chord) > 1:
        minFret, maxFret = None, None
        for note in chord:
            if note[0] > 0 :
                if maxFret == None or note[0] > maxFret:
                    maxFret = note[0]
                if minFret == None or note[0] < minFret:
                    minFret = note[0]

        if minFret == None or maxFret == None:
            return 0
        else:
            return maxFret - minFret
    else:
        return 0

def averageFretNumber(individual):
    '''
    for a given tablature returns an average of all fret positions
    '''
    noteCounter = 0
    fretPositionSum = 0.0
    for timeSlot in individual:

        noteCounter += len(timeSlot)
        for note in timeSlot:
            fretPositionSum += note[0]
    #return average

    return 0 if noteCounter == 0 else fretPositionSum / noteCounter


def averageSubsequentFretDistance(individual):
    '''
    for a given tablature returns an average distance between subsequen frets
positions
    '''
    timeSlotCounter = 0
```

```
    fretPositionDistanceSum = 0.0
    lastFretPosition = None
    for timeSlot in individual:

        #open string positins will not contribute to the average
        if timeSlot[0][0] != 0:
            #consider only first note
            if lastFretPosition == None:
                lastFretPosition = timeSlot[0][0]
            else:
                timeSlotCounter += 1
                fretPositionDistanceSum += abs(lastFretPosition - timeSlot[0][0])
                lastFretPosition = timeSlot[0][0]

    #return average
    return 0 if timeSlotCounter == 0 else fretPositionDistanceSum / timeSlotCounter


def tabMultiFitness(individual):
    '''
    for a given tablature returns a tuple representing a multiobjective fitness
    '''

    return averageFretNumber(individual), averageSubsequentFretDistance(individual)


def tabMutate(individual, indpb):
    '''
    performs mutation on the individual. Each chord in the score will be altered with
    probability of indpb.
    '''
    for indx in xrange(len(individual)):
        if random() < indpb:

            timeSlotLength = len(individual[indx])

            allFingerings = None

            if timeSlotLength == 1:
                fingering = individual[indx][0]
                noteTuple = convertFingeringTupleToNoteTuple(fingering)
                allFingerings = getAllFingerings(noteTuple)
            else:
                chordFingering = individual[indx]
                chordNotes = list()
                for fingering in chordFingering:
                    chordNotes.append(convertFingeringTupleToNoteTuple(fingering))
                allFingerings = getValidChordsForFingerings(chordNotes)

            #if this is the only possible fingering, then nothing to mutate
            if len(allFingerings) > 1 :
                #simulate do while loop
                doWhileCondition = True
                while doWhileCondition:
                    randomFingering = choice(allFingerings)
                    if timeSlotLength == 1:
                        if (randomFingering != fingering):
                            doWhileCondition = False
                            individual[indx][0] = randomFingering
                    else:
                        if len(set(chordFingering) - set(randomFingering)) != 0:
                            doWhileCondition = False
                            individual[indx] = randomFingering


    return individual,

def convertFingeringTupleToNoteTuple(fingeringTuple):
    '''
```

```
    converts a given tablature fingering back to note in a classical notation
representation
    '''

    #calculating starts of C3, for each string define offset in half notes
    stringOffsets = (4,9,14,19,23,28)

    offsetFromC3 = fingeringTuple[0] + stringOffsets[ fingeringTuple[1] ]

    offsetOctave = offsetFromC3 / 12

    oneOctaveOffset = offsetFromC3 % 12

    noteMapping = ('C',
                   'C',
                   'D',
                   'D',
                   'E',
                   'F',
                   'F',
                   'G',
                   'G',
                   'A',
                   'A',
                   'B'
                  )

    accidental = 'n'

    if oneOctaveOffset in (1,3,6,8,10):
        accidental = '#'


    return (noteMapping[oneOctaveOffset], accidental, 3 + offsetOctave)



def printTablature(tab, widthMax=None):
    '''
    prints a tablature to standard output in a human readable form
    the widthMax argument can be used to limit width of the output to 'widthMax' notes
per line
    each note takes three characters
    '''
    stringNames = ("E","A","D","G","B","E")

    if (widthMax == None or widthMax > len(tab)):
        widthMax = len(tab)

    i = 0
    while(i*widthMax < len(tab)):
        tabSlice = tab[i*widthMax:i*widthMax+widthMax]
        i = 1 + i
        #print one line of tablature at a time
        for stringNo in range(5,-1,-1):
            sys.stdout.write(stringNames[stringNo] + "--")

            for notes in tabSlice:
                found = False
                for note in notes:
                    if note[1] == stringNo:
                        found = True
                        sys.stdout.write("-" + str(note[0]))
                        if note[0] < 10:
                            sys.stdout.write("-")
                if found == False:
                    sys.stdout.write("---")
            print ""
```

```
        print ""


def main():
    pass


if __name__ == "__main__":
    main()
```

8.3.3 input.py

Note that for brevity, only one score from this source file is fully presented here. The remaining 3 can be found in the actual sources that are a part of this report.

```
# Author: Artur Jablonski
# Date: Sept 2012
#
# This software is submitted in partial fulfillment of the requirements for the
# BSc (Honours) Information Systems and Information Technology (DT249)
# to the School of Computing, Faculty of Science, Dublin Institute of Technology.
'''
This module contains manually converted music scores to
internal representation.

The scores were used for testing the algorithm in gentab.py
'''

cMajorScale = [[('C','n', 4)],
               [('D','n', 4)],
               [('E','n', 4)],
               [('F','n', 4)],
               [('G','n', 4)],
               [('A','n', 4)],
               [('B','n', 4)],
               [('C','n', 5)],
               [('D','n', 5)],
               [('E','n', 5)],
               [('F','n', 5)],
               [('G','n', 5)],
               [('A','n', 5)],
               [('B','n', 5)],
               [('C','n', 6)]
              ]
```

8.3.4 TestCore.py

```
# Author: Artur Jablonski
# Date: Sept 2012
#
# This software is submitted in partial fulfillment of the requirements for the
# BSc (Honours) Information Systems and Information Technology (DT249)
# to the School of Computing, Faculty of Science, Dublin Institute of Technology.

'''
TestCore.py

This class is a unit test suite for the core.py module
'''
import unittest
import core


class Test(unittest.TestCase):
    '''
    This class is a unit test suite for
    functions defined in core.py
    '''

    def testGetValidChordsForFingerings(self):
        #empty chord
        result = core.getValidChordsForFingerings([])
        self.assertTrue(len(result) == 0)

        cMajorChord = [('C','n',4),('E','n',4),('G','n',4)]
        result = core.getValidChordsForFingerings(cMajorChord)
        self.assertTrue(len(result) == 3)
        self.assertTrue(((8, 0), (7, 1), (5, 2)) in result)
        self.assertTrue(((3, 1), (2, 2), (0, 3)) in result)
        self.assertTrue(((8, 0), (7, 1), (0, 3)) in result)


        #test 6 string chord
        A7 = [('A','n',3),('E','n',4),('G','n',4),('C','#',5),('E','n',5),('A','n',5)]
        result = core.getValidChordsForFingerings(A7)
        self.assertTrue(len(result) == 2)
        self.assertTrue(((5, 0), (7, 1), (5, 2), (6, 3), (5, 4), (5, 5)) in result)
        #this is unexpected but valid fingering!
        self.assertTrue(((0, 1), (12, 0), (0, 3), (11, 2), (0, 5), (10, 4)) in result)

    def testInitTablature(self):
        #empty
        result = core.initTablature(list, [])
        self.assertTrue(len(result) == 0)
        cMajorScale = [[('C','n', 4)],[('D','n', 4)],[('E','n', 4)],[('F','n', 4)],
[('G','n', 4)],
                       [('A','n', 4)],[('B','n', 4)],[('C','n', 5)],[('D','n', 5)],[('E','n',
5)],[('F','n', 5)],
                       [('G','n', 5)],[('A','n', 5)],[('B','n', 5)],[('C','n', 6)]
                      ]
        result = core.initTablature(list, cMajorScale)
        #what a great place to use list comprehensions and even lambda form!
        convertFingeringTupleChordToNoteTupleChord = lambda x:
[core.convertFingeringTupleToNoteTuple(fingeringTuple) for fingeringTuple in x]
        backToClassicalScore = [convertFingeringTupleChordToNoteTupleChord(x) for x in
result]

        assert(cMajorScale == backToClassicalScore)

        randomNotes = [[('C','n', 4),('A','n', 4),('E','n', 4)],[('G','n', 6)],
[('E','n', 4),('F','n', 4),('G','n', 5)]]
```

```
        result = core.initTablature(list, randomNotes)

        backToClassicalScore = backToClassicalScore =
[convertFingeringTupleChordToNoteTupleChord(x) for x in result]

        assert(randomNotes == backToClassicalScore)

    def testGetAllFingerings(self):
        #empty
        result = core.getAllFingerings(list())
        self.assertTrue(len(result) == 0)

        #one fingering
        result = core.getAllFingerings(('F', '#', 3))
        self.assertTrue(len(result) == 1)
        self.assertTrue((2,0) in result)

        #two fingerings
        result = core.getAllFingerings(('C', 'n', 4))
        self.assertTrue(len(result) == 2)
        self.assertTrue((8,0) in result)
        self.assertTrue((3,1) in result)

        #six fingerings
        result = core.getAllFingerings(('E', 'n', 5))
        self.assertTrue(len(result) == 6)
        self.assertTrue((24,0) in result)
        self.assertTrue((19,1) in result)
        self.assertTrue((14,2) in result)
        self.assertTrue((9,3) in result)
        self.assertTrue((5,4) in result)
        self.assertTrue((0,5) in result)

        #out of guitar range
        result = core.getAllFingerings(('D', '#', 3))
        self.assertTrue(len(result) == 0)
        result = core.getAllFingerings(('F', 'n', 7))
        self.assertTrue(len(result) == 0)

    def testGetChordFretSpan(self):
        #empty
        result = core.getChordFretSpan(list())
        self.assertTrue(result == 0)

        result = core.getChordFretSpan([(3,3)])
        self.assertTrue(result == 0)

        result = core.getChordFretSpan([(3,3),(5,5)])
        self.assertTrue(result == 2)

        result = core.getChordFretSpan([(3,3),(5,5),(0,2),(0,1)])
        self.assertTrue(result == 2)

        result = core.getChordFretSpan([(3,3),(5,5),(1,2),(0,1)])
        self.assertTrue(result == 4)

    def testAverageFretNumber(self):
        #empty
        result = core.averageFretNumber(list())
        self.assertTrue(result == 0)

        result = core.averageFretNumber([[(3,3)]])
        self.assertTrue(result == 3)

        result = core.averageFretNumber([[(3,3)],[(5,5)]])
        self.assertTrue(result == 4)

        result = core.averageFretNumber([[(3,3),(5,5),(0,2),(0,2)]])
        self.assertTrue(result == 2)
```

```
        result = core.averageFretNumber([[(3,3),(5,5),(1,2),(7,1)]])
        self.assertTrue(result == 4)

    def testAverageSubsequentFretDistance(self):
        #empty
        result = core.averageSubsequentFretDistance(list())
        self.assertTrue(result == 0)

        result = core.averageSubsequentFretDistance([[(3,3)]])
        self.assertTrue(result == 0)

        result = core.averageSubsequentFretDistance([[(0,3)]])
        self.assertTrue(result == 0)

        result = core.averageSubsequentFretDistance([[(3,3)],[(5,5)]])
        self.assertTrue(result == 2)

        result = core.averageSubsequentFretDistance([[(3,3)],[(5,5)],[(0,4)]])
        self.assertTrue(result == 2)

        result = core.averageSubsequentFretDistance([[(3,3)],[(5,5)],[(0,2)],[(0,2)]])
        self.assertTrue(result == 2)

        result = core.averageSubsequentFretDistance([[(3,3)],[(5,5)],[(1,2)],[(7,1),
(4,5)]])
        self.assertTrue(result == 4)

    def testTabMultiFitness(self):
        tab = [[(2,3),(4,3)],[(5,2)],[(3,3)],[(5,4)]]
        afn, asfd = core.averageFretNumber(tab),
core.averageSubsequentFretDistance(tab)
        result = core.tabMultiFitness(tab)
        self.assertTrue(result == (afn, asfd))

    def testTabMutate(self):


        tab = [[(2,3),(4,3)],[(5,2)],[(3,3)],[(5,4)]]
        tabCopy = tab[:]

        core.tabMutate(tab, 1)

        #the mutant must be different, but must translate to the same music
        self.assertTrue(tab != tabCopy)

        convertFingeringTupleChordToNoteTupleChord = lambda x:
[core.convertFingeringTupleToNoteTuple(fingeringTuple) for fingeringTuple in x]

        backToClassicalScoreTab = [convertFingeringTupleChordToNoteTupleChord(x) for x
in tab]
        backToClassicalScoreTabCopy = [convertFingeringTupleChordToNoteTupleChord(x)
for x in tabCopy]

        self.assertTrue(backToClassicalScoreTab == backToClassicalScoreTabCopy)


    def testConvertFingeringTupleToNoteTuple(self):

        result = core.convertFingeringTupleToNoteTuple((3,4))
        self.assertTrue(result == ('D','n',5))

        result = core.convertFingeringTupleToNoteTuple((0,0))
        self.assertTrue(result == ('E','n',3))

        result = core.convertFingeringTupleToNoteTuple((24,5))
        self.assertTrue(result == ('E','n',7))
```

```
if __name__ == "__main__":
    #import sys;sys.argv = ['', 'Test.testName']
    unittest.main()
```