Progress Report

John Pederson

registration: 100266127

1 Introduction

Guitar tablature is used by many guitarist as a way to transcribe a piece of music that is read and played by anyone, even by those unable to read standard notation. Normally this is done by skilled guitarists, but automating the process would allow anyone to create tablature, lowering the skill floor for reading guitar music and opening it up to more people. The aim for this project is to create a program which, when given an MIDI file input, will produce guitar tablature that is both accurate to the original piece of music and not unnecessarily complex to play. This report will cover the progress made so far and the plan for the project moving forward.

2 Research and study

2.1 Finding a method

The first task was deciding on a method that would provide good results within the time-frame for the project. Researching the topic revealed many different methods that have been used in the past. The first few weeks were spent looking into which methods would suit this project the best, finally coming to a choice between genetic algorithms (like the one used in Tuohy and Potter (2005)) and neural networks (after some inspiration by Wiggins and Kim (2019)).

Genetic algorithms model evolution over generations to find optimal solutions for the given problem. First a fitness function is defined, which provides a metric for measuring how optimal a particular solution is. A random selection of solutions are taken as a starting point (the first generation), then the most optimal of these solutions are taken and combined by a crossover function, creating a new generation as the result. This is repeated cyclically for the desired number of generations. This method of combining the most optimal solutions models creatures selecting the best available mate for procreation during the evolution process. For a more in depth explanation of genetic algorithm,s see Dockhorn and Lucas (2022). This method is relatively simple to implement and has been shown to produce strong results (once again see Tuohy and Potter (2005)).

Neural networks are machine learning methods which model activity in the brain, using layers of 'neurons' to produce a desired output from the given input. On a basic level each neuron acts as an independent algorithm, taking input from either other neurons or

the initial input depending on if it is in a surface or internal (hidden) layer. The outputs from each neuron are then fed into the next layer until the output layer is reached. There are many different types of neural network with varying relevance to this project. Chen (2022) has a much more detailed explanation of the topic with examples of some of the different types of neural network. Neural networks are a powerful too, however they can be incredibly time-restrictive and internal workings may quickly become increasingly enigmatic as the network learns.

After researching these methods the genetic algorithm seemed most suitable for this project. The time constraints simply make attempting to develop a functional neural too much of a risk.

2.2 Studying genetic algorithms and the 8 queens problem

Once a genetic algorithm was decided upon, the next step was to become proficient in the development and use of genetic algorithms. To this end development was started on an genetic algorithm that would provide solutions to the well known 8 queens problem (arranging 8 queens on an 8x8 chess board so that none of them can take any of the others). Genetic algorithms have been used many times to solve the 8 queens problem (even having the 8 queens used as an example to explain genetic algorithms by Dockhorn and Lucas (2022)) and therefore due to the abundance of documentation on the topic and the relatively simple nature of the problem it was an obvious choice for exploring the practical implementation of the algorithm.

2.2.1 Implementing the algorithm

The first step to creating the algorithm was deciding how to represent the chessboard in code. This is an important step as it allows the search space to be minimised by encoding the chessboard to reduce the number of non-accurate solutions (e.g., solutions containing more less than 8 queens etc.). To this end the chessboard was encoded as an array with eight elements, each element representing a queen with the value of the element as the row number and the index as the column number. With the values one to eight then filled in in a random order. This method ensures that the solution will have 8 queens, none of which share a row or a column. Therefore the only remaining problem is queens sharing a diagonal.

Once the board was encoded a fitness function was needed to identify which chessboards were closest to optimal. This was a simple matter of counting the number of queens in the same diagonal. The board with the lowest number of attacking queens is then deemed the most optimal. Due to the way this function was coded, if there are more than two queens in a single diagonal then any queens over the second will be counted multiple times as attacking queens. This was not intentional and was merely a result of the code, however it was left in as having more than two queens in a single diagonal is arguably further from the desired solution.

After defining the fitness function a crossover method was used to pair up the current generation and combine them into children for the next generation. In this crossover function the current generation is sorted by the fitness value calculated in the fitness function (lowest first as this denotes a more desirable solution). The most optimal solutions were then paired up (the first paired with the second, third with the fourth etc.) and the least optimal were also paired to maintain population size. For each pair a random selection of the queens are selected to make two children, maintaining population size. This is done in a way that minimises the possibility of the children containing duplicate rows, however, if the children do end up with duplicates, the crossover method replaces these with any rows that are missing from the child. This replacement is similar to mutations used by most genetic algorithms except for the fact that it is not as inherently random as mutations generally are.

2.2.2 Experimenting with the algorithm

In order to find the optimal parameters, the algorithm was tested on different population sizes from 1000 to 5000 (around $\frac{1}{20}$ to $\frac{1}{8}$ of the total search space) and for different numbers of generations between 20 and 100. Each of these configurations was tested on 100 different random populations. The results were measured using a number of metrics. The number of 'ideal' solutions (those with no attacking queens) is the most valuable metric as this is the entire goal of the algorithm, this was measured by the difference from the starting population rather than the total number of ideal solutions due to the small chance of an ideal solution being in the random selection for the starting population. The average change in fitness was taken to see how the overall population improved (for this metric negative numbers were preferable as the lower the fitness the more optimal the solution). However, due to the least optimal solutions also being paired

together in the crossover, the bottom end of the population more often than not became less optimal rather than more. Therefore the average change in fitness for the top 10% of the population was also taken as this is the area that the algorithm is focused on. The results of these tests can be found in Figure 1.

As can be seen in the table, every configuration of the algorithm provided at least one ideal solution so in every case the algorithm was successful in solving the 8 queens problem, though the number of solutions found improved with both the number of generations and the population size as could be expected. Also as expected, the crossover of the least fit solutions caused the average fitness to increase. More surprisingly the average fitness of the top 10% of the population, while lower, was not significantly improved. This implies that a large amount of the improvement is seen over a very small section of the population. The change in fitness was largely the same for each population size, with the change in number of generations showing the only notable difference. This shows that the population size has little effect on the performance of the algorithm, likely only increasing the found solutions due to a larger number of solutions that are already nearly in an ideal state. It may also be worth it to take the top `n` number of solutions for each set of parameters and compare the fitness improvement for those as this may give more information than the top percentage of the solutions, however this was not done by the time of this report due to time constraints.

2.2.3 Application to tablature transcription

Working on the 8 queens problem has provided a lot of insight into genetic algorithms and how they function when applied to a specific problem. Moving into the tablature transcription there are a few specific things that can be considered from these results. If the algorithm performs similarly on each population size it may be more important to find the lowest population required for the algorithm to produce an acceptable result, rather than finding an 'optimal' population as was attempted here. Due to the lower end of the population becoming increasingly unfit, it may be more viable to introduce new random members to the population in place of the lower set at certain points, which should improve performance of the algorithm.

3 Problems and adjustments

The initial plan included having a working prototype by week 12, but a large portion of the time allocated to developing the prototype was taken up by the research and selection of a development method and the subsequent study of genetic algorithms. This has led to the project being significantly behind the initially proposed schedule. In order to remedy this a new, more thorough, plan has been created which will allow goals to be hit more reliably due to less time spent working out the next steps.

4 Moving forward

4.1 Project plan

Due to the significant departure from the original project plan, a new Gantt chart (Figure 2) has been created outlining the updated project plan, with more detail added due to more concrete understanding of the proposed system. The aim is to have the system producing tablature as soon as possible, then to iteratively improve upon the algorithm to reach optimal performance. The submission date has not been officially set so the final deadline for the project plan is the 15th of May as this is the start of the last week of the semester.

4.2 System structure

A provisional class diagram can be seen in Figure 3, there will be a specific class for each note, containing information such where on the fretboard the note can be played and how long the note will last. These will be used when generating tablature objects, which will contain the notes either on their own or combined into chord objects and will also have a variable for the tablature's fitness calculated by a method inside the tablature class. The transcription algorithm will take the notes read from the MIDI file, encode them into note objects and create possible tablatures from these objects for the starting population. The algorithm will then run a set number of generations before sending the most fit tablature to be transcribed with the TAB software.

Population Size	No. Generations	Average No. of Additional Ideal Solutions	Average Change in Fitness	Average Change in Fitness (Top 10%)
1000	20	1.3	0.13798	-0.0287
1000	40	1.23	0.1975	-0.0279
1000	60	1.4	0.26002	-0.0463
1000	80	1.64	0.28739	-0.0305
1000	100	2.06	0.32674	-0.0501
2000	20	1.64	0.13597	-0.0372
2000	40	2.27	0.195575	-0.0341
2000	60	2.5	0.264315	-0.03835
2000	80	3.18	0.313175	-0.0507
2000	100	5.13	0.38225	-0.0618
3000	20	1.69	0.15257	-0.022366667
3000	40	3.58	0.214856667	-0.037033333
3000	60	3.88	0.278033333	-0.0366
3000	80	5.19	0.35111	-0.031366667
3000	100	6.48	0.397283333	-0.053266667
4000	20	3.24	0.1489275	-0.0357
4000	40	3.69	0.23222	-0.030725
4000	60	4.84	0.2868625	-0.024125
4000	80	6.87	0.3573	-0.04185
4000	100	8.99	0.40729	-0.066675
5000	20	2.71	0.146336	-0.02304
5000	40	5.1	0.231152	-0.01426
5000	60	7.55	0.286868	-0.04766
5000	80	8.31	0.377294	-0.04032
5000	100	11.6	0.435136	-0.05804

Figure 1: Results of 8 queens genetic algorithm testing

References

Chen, J. (2022). What is a neural network? *https://www.investopedia.com/terms/n/neuralnetwork.asp*.

Dockhorn, A. and Lucas, S. (2022). Choosing representation, mutation, and crossover in genetic algorithms. *IEEE Computational Intelligence Magazine*, 17(4):52–53.

Tuohy, D. R. and Potter, W. D. (2005). A genetic algorithm for the automatic generation of playable guitar tablature. In *ICMC*, pages 499–502.

Wiggins, A. and Kim, Y. E. (2019). Guitar tablature estimation with a convolutional neural network. In *ISMIR*, pages 284–291.

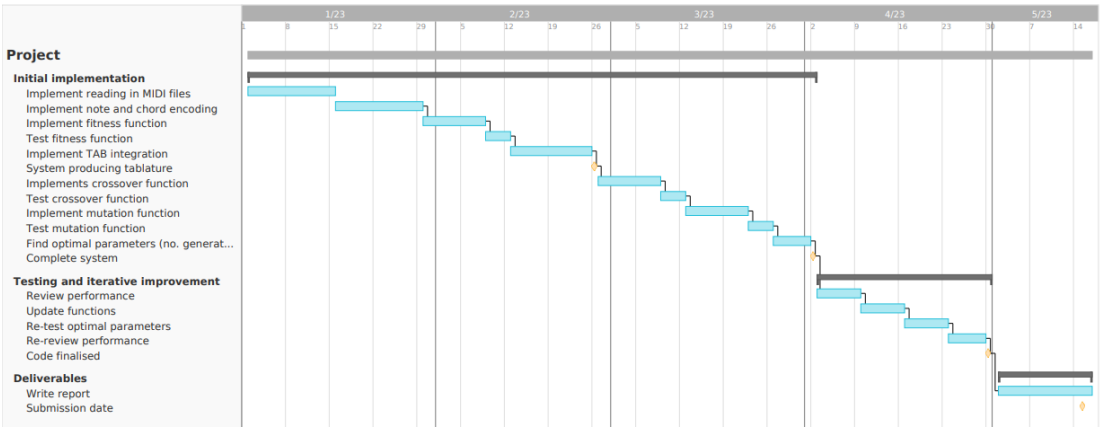

Figure 2: Updated Gantt chart, created using *www.teamgantt.com*

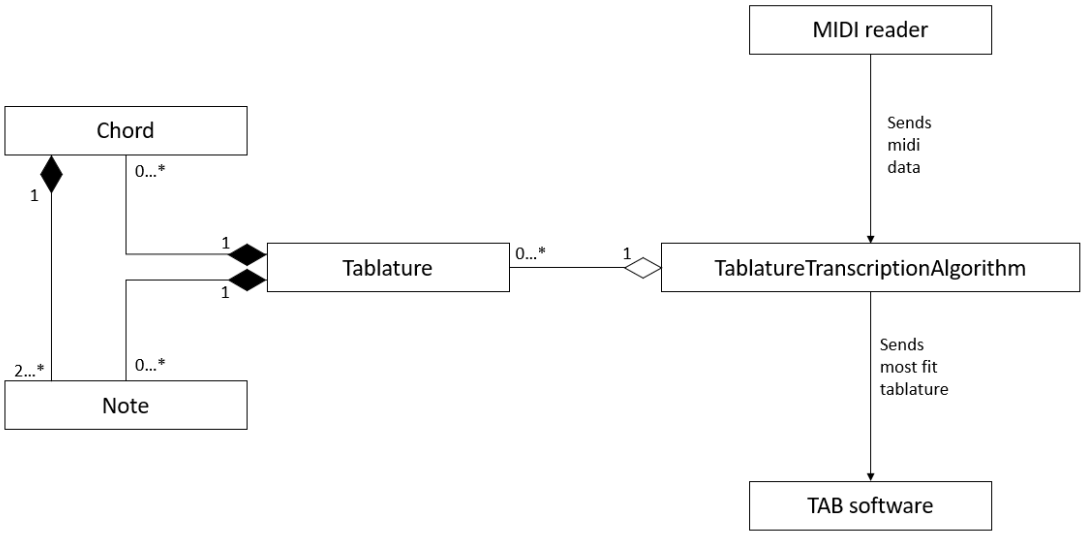

Figure 3: Provisional class diagram