Computing Guitar Fingerings

Niki Stavropoulou
s1641718

Progress Report

The project's aim is to develop a framework that can generate and compute playable left hand guitar fingerings, given a music piece. The task is divided in the computation of fingerings for single note sequences and fingerings for chords. For the fingering computation and evaluation custom rules will be used that are based on guitar playing experience. This is a problem in itself, since there is no actual way of defining the 'best' fingering; it can vary given a person's expertise, experience and, most importantly, personal comfort and preference. I will describe briefly the progress I made.

Initially, I wrote code to read and parse a music piece in MusicXML format, using python 3.5. I extract information from XML elements and use it to create a tree-like structure of the notes in the file. The code uses two imported files, one that contains the rule weights, the importance of which I will refer on later, and one that contains information about all note placements. The latter contains all the possible placements (fret-string combinations) for all the notes, both accidentals and naturals, that can be played on the first twelve frets of a guitar. Using these, the program can further compute all the finger-note-fret-string combinations for each note that is being parsed.

More specifically, given the note sequence parsed, the code expands the tree starting from the first note till the last one, creating nodes for each possible finger the notes can be played with. Each node holds additional information, like its parent note node, its note pitch and octave, and a score. The score is computed taking into account information like the fret-finger distance and adds to the node's parent's score.

After the tree has expanded for all notes in the initial music file, the program can compute the path with the fingerings that has the best score. The output is a new MusicXML file that contains the computed fingerings.

At first the code was expanding the tree treating all notes in the music file as a sequence. I made a change while creating the tree, so as to cover both note sequences and note chords. A tree node now represents a chord, that can either have one note or multiple ones. In the latter case, the node computes the appropriate fingering combinations for the chord and keeps the information for all the notes in the chord, before proceeding with the tree expansion. The same scoring as for sequences is used. The single note sequence is been processed as before. An example of the program's output is shown in the following pictures.

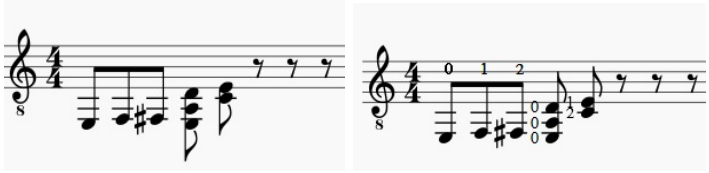

Figure 1: An example containing both sequences and chords and its computed fingerings.

The size of the tree proved to be a big problem, so a custom form of pruning was needed. While a simple toy sequence of 4 notes could be processed fast, the first measure of 'Dust in the wind' created 8398080 leaf nodes, since it expanded for all possible fret-string-finger combinations. To facilitate memory usage and the overall performance I removed any recursions from the code and rewrote the tree implementation to further limit the width while expanding. The first limitation is applied when considering how many children to add to a node. The code will only allow adding a set number of children. The second limitation is applied at thresholds while parsing, meaning,

for example every 10 notes, the width of the leafs is reduced to a minimum, before continuing with expanding. This reduces the overall search space.

Finally, the most important aspect of the project it the way the fingerings are scored. The code scores the fingerings according to custom fingering rules. What this means it that the code evaluates a custom representation of information for the current finger combination. This representation relies on significant fingering characteristics, chosen based on guitar playing experience, such as a change in position, whether the current note is played on an open string or using the same finger consecutively. The score is computed by penalizing or favoring these characteristics. For this computation, I use the imported file with the weights corresponding to each characteristic; the higher the node score, the less optimal the node fingering.

To be able to make changes to the rules easier, I created a graphical interface using python's Tkinter. The user can input the file for which it wants to compute the fingerings, but also see the rules that are been used and change their impact (their corresponding weight for the computations, which is a numerical value) or add new ones.

Currently, I am experimenting with applying different rules and noting on their importance in producing playable fingerings. The feasibility of my program's results on toy examples are firstly evaluated by me. For some music pieces, I also compare the code's output with fingerings that other guitar players suggested and already existing ones, and discuss the performance of my framework. Given the nature of the project, the results were not expected to be perfect, since we are trying to simulate a guitar player's way of thinking and map experience to code. Nevertheless, the majority of computed fingerings on the smaller examples so far were playable - acceptable, if not optimal. For the rest of the project's duration, I will proceed to further refine the framework's rules, so as to produce better results.