A SIMPLE ALGORITHM FOR AUTOMATIC GENERATION OF POLYPHONIC PIANO FINGERINGS

Alia Al Kasimi
Dept. of Computer Science
Indiana Univ.
alia.alkasimi@gmail.com

Eric Nichols
Dept. Of Computer Science
Indiana Univ.
epnichol@indiana.edu

Christopher Raphael
School of Informatics
Indiana Univ.
craphael@indiana.edu

ABSTRACT

We present a novel method for assigning fingers to notes in a polyphonic piano score. Such a mapping (called a "fingering") is of great use to performers. To accommodate performers' unique hand shapes and sizes, our method relies on a simple, user-adjustable cost function. We use dynamic programming to search the space of all possible fingerings for the optimal fingering under this cost function. Despite the simplicity of the algorithm we achieve reasonable and useful results.

1. INTRODUCTION

Several different algorithms have been proposed in recent decades for automatic generation of musical instrument fingerings [1,2,3,4,5]. We propose to build on this previous work in two ways. First, in keeping with the nature of most piano music, our algorithm handles *polyphonic* input – the lack of a polyphonic algorithm in the literature is lamented in [2]. Second, we formulate the solution in an elegant manner that facilitates rapid implementation and computation.

2. AUTOMATIC FINGERING ALGORITHM

2.1. Representation of Solution Space

Beginning with a symbolic representation of a musical score, we generate a trellis graph[1] with one layer for each time point where a note begins or ends, and for each layer a set of nodes representing all possible fingerings for the notes sounding at that time point.

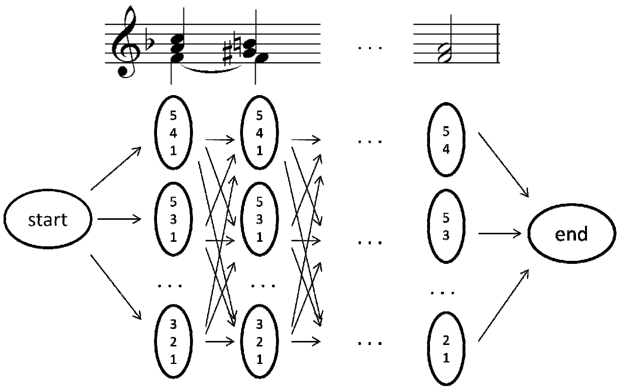

Figure 1. Trellis Graph of Solution Space

[1] A trellis graph is a multilayer directed acyclic graph where nodes are only connected between adjacent layers.

The numbers correspond to fingers, where '1' represents the thumb and '5' is the little finger. We constrain the possible fingerings as follows:

1. Within each node, fingers are assigned in order from lowest pitch to highest pitch (no crossover).
2. The same finger must be used as long as a note persists (no finger substitution is allowed).
3. Each finger may only depress one note.

Each path from 'start' to the 'end' is a possible fingering solution. A weight (cost) is assigned to each connection between nodes in the graph, and the Viterbi algorithm is used to compute the optimal fingering sequence.

Previous monophonic fingering approaches have used a cost function based on the stretch in the hand between two notes. We call this the **horizontal cost** and extend it to the polyphonic case. Additionally, we introduce the **vertical cost** quantifying the spread of the fingers involved in playing a particular chord. The cost of transitioning between two nodes is the sum of the horizontal and vertical costs.

2.2. Vertical Cost

The vertical cost depends on the combination of pitches that are present at a given point in time. Note that we can always rewrite polyphonic music as a collection of chords by using the tie symbol (Figures 2 and 3):

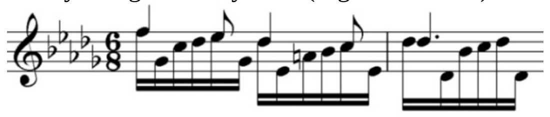

Figure 2. Excerpt from Aufschwung (*Phantasiestuck* op.12) by Schumann

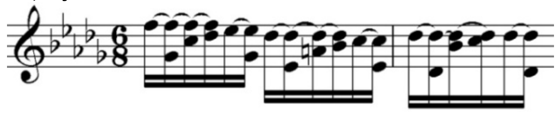

Figure 3. Vertical analysis of the same excerpt

To compute the vertical cost of a node, we first decompose the fingering in the node into adjacent pairs of notes. For example, in the F Major triad *F-A-C* with associated fingering 1-3-5, we consider the two pairs *F-A* (1-3) and *A-C* (3-5). Each pair's cost is determined by which two fingers are involved and the distance between the two notes being played, yielding the node cost:

$$Cost_v(n_i) = \sum_{k=1}^{\#pairs} c_v(P_k, d) \quad (1)$$

$c_v(p,d)$ gives the cost for playing two notes separated by distance d with a particular pair p of fingers.

2.3. Horizontal Cost

The horizontal cost between two nodes is defined to be the average transition cost taking into account all pairs of notes consisting of one note from each chord:

$$Cost_h(n_a \rightarrow n_b) = \frac{1}{N} \sum_{i \in n_a, j \in n_b} c_h(i \rightarrow j, d) \qquad (2)$$

Specifically, in Figure 4, the horizontal cost $c_h(p,d)$ is the sum of the costs for transitioning from *C* to *F*, *C* to *A*, *E* to *F*, *E* to *A*, *G* to *F*, and *G* to *A*, divided by 6.

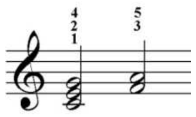

Figure 4. Horizontal cost example

One might expect that c_h and c_v are equivalent – after all, playing an ascending third (e.g. *C* up to *E*) with fingers 1 and 3 is just as easy as playing *C* and *E* simultaneously. However, there are two situations where this is not true. First, hand motion is allowed between the notes in horizontal cost. Second, if the direction of motion is reversed, a *crossover* occurs and difficulty increases.

In our implementation we ignore the first of these problems, and set c_h=c_v. In the case of a crossover situation, however, c_h is defined by a separate function. Crossovers involving the thumb are assigned lower cost.

2.4. User Specification of Cost Functions

In order to generate fingerings tailored to a specific user, we ask the user a series of questions about the difficulty of playing various intervals with particular pairs of fingers. The responses are used to define cost functions c_v and c_h for each pair of fingers. Each cost function has the same general shape as the example in Figure 5, which represents all possible costs of stretches involving fingers 2 and 5. In this example, the user has specified the following information:

- The smallest interval that is easily playable by fingers 2 and 5 is 4 half steps (smaller intervals result in uncomfortable compression of the hand).
- The comfort zone lies between 4 and 7 half steps.
- After eleven half steps the stretch is nearly impossible.

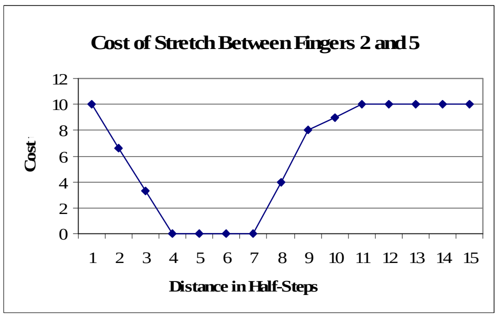

Figure 5. Cost function c_v for finger pair 2,5

3. RESULTS

We tested our system on several different music excerpts and considering the simplicity of the approach we are pleased that this algorithm generates quite plausible polyphonic piano fingerings. Additional improvements would include allowing finger substitution during a sustained note and modelling the difference between white and black keys. We also would like to extend the system to determine which hand (left or right) should play each note. See Figure 6 for sample output.

4. REFERENCES

[1] Hart, M., Bosch, R., & Tsai, E. (2000). "Finding Optimal Piano Fingerings." *The UMAP Journal, 21*(2), 167-177.

[2] Lin, C. And Liu, D. "An Intelligent Virtual Piano Tutor". *Proceedings of VRCIA 2006*. 353-356. Hong Kong, June 14-17, 2006.

[3] Parncutt, R., Sloboda, J. A., Clarke, E. F., Raekallio, M., & Desain, P. (1997). "An Ergonomic Model of Keyboard Fingering for Melodic Fragments". *Music Perception, 14*(4), 341-382.

[4] Radicioni et. al. "A segmentation-based prototype to compute string instruments fingering". *Proceedings of the Conference of Interdisciplinary Musicology (CIM04).* Graz, Austria, April 15-18, 2004.

[5] Sayegh, S. (1989). "Fingering for String Instruments with the Optimal Path Paradigm". *Computer Music Journal. 13*(3), 76-84.

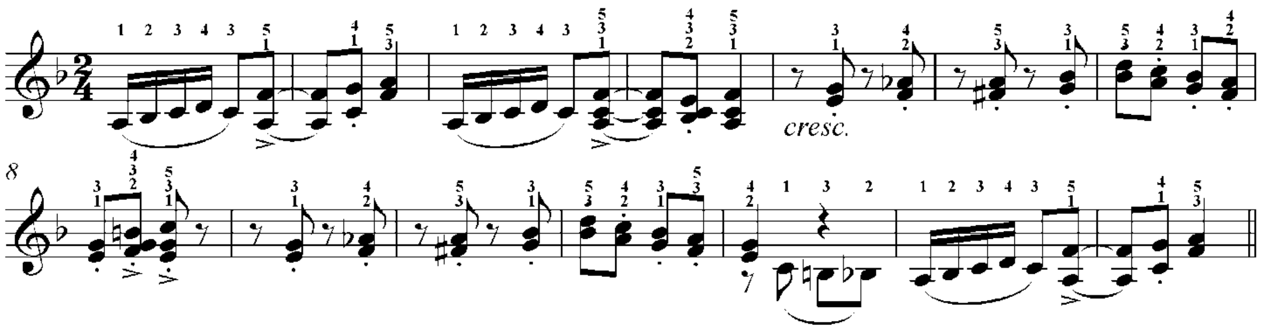

Figure 6. System output: Excerpt from Ellmenreich's Spinning Song