KTH Computer Science
and Communication

Suggested fingering for keyboards

BASSAM ALFARHAN
DAVID SANDBERG

Bachelor of Science Thesis
Supervisor: Anders Askenfelt
Examiner: Mårten Björkman

Stockholm, Sweden 2013

Abstract

Learning to play the piano can be a difficult task since it is often not clear which finger should play which key on the keyboard. An algorithm that provides the pianist with a suggested fingering for a piece of music would therefore aid in learning to play the piano as efficiently as possible. All the algorithms developed to date have made use of different rules to calculate the difficulty of a specific fingering. The aim of this study was to take some of these rules and add additional functionality in order to create an algorithm that was able to produce reasonable fingerings.

The question was if it was possible to add support for interleaved triads and different articulations to the set of rules and create a reasonable algorithm.

The results confirmed that it was possible but that further development would be needed in order to prove that the added functionality was consistent.

Sammanfattning

Att lära sig spela piano kan vara svårt eftersom det är oklart vilken not som skall spelas med vilket finger. En algoritm som föreslår en lämplig fingersättning för ett musikstycke skulle därför kunna underlätta inlärningen. Alla algoritmer utvecklade hittills har använt sig av olika regler för att räkna ut en viss fingersättnings svårighetsgrad. Målet med denna studie var att använda några av dessa regler och lägga till extra funktionalitet för att skapa en algoritm som kunde producera lämpliga fingersättningar.

Frågan var om det var möjligt att addera stöd för treklanger och olika artikulationer till reglerna och skapa en användbar algoritm.

Resultaten visade att detta var möjligt men att mer utvärdering behövs för att kunna bevisa att algoritmen fungerar för alla fall.

Statement of collaboration

Bassam was responsible for implementing the algorithm in Java code as well as testing and debugging the proof of concept. David was responsible for the collaboration with the pianist and structuring the essay.

The majority of the essay was written by both authors. However, the sections where one of the authors has contributed to a greater extend can be found in the table below.

Section	Main contributor
Abstract, Sammanfattning, Introduction, Background	David
Method, Discussion	Bassam

Contents

Chapter 1

Introduction

Learning to play an instrument can be an extremely difficult task. Not only are you expected to understand how to read musical notation, but there are also other aspects to be taken into account, such as how fast a sequence should be played and what kind of intonation to use. There is also the problem of deciding which finger plays which note. For instruments such as woodwinds (clarinet, saxophone etc.) the fingering is usually predetermined due to the instrument design. On the clarinet for example, it is ergonomically inconvenient to play the key closest to the bell with one of your thumbs. To reach all the keys the fingers may have to stretch but no hand movement is needed, and each finger is mapped to a certain key. For people learning to play woodwinds, the question of fingering is not a problem. The same cannot be said about the piano where there is no standard fingering for a certain note. In this case the "optimal fingering of a note depends [...] almost entirely on the context- both physical (on the keyboard) and musical (as expressed in the score) - in which it appears." (Parncutt, Sloboda, Clarke, Raekallio and Desain 1997).

Experienced pianists usually have their own idea of which fingering works best for a given piece of music. The fingering they choose might depend on the musical articulation (e.g. legato, staccato, marcato), tempo or physiological constraints among other properties. For less experienced pianists, however, the choice of fingering might not be obvious and an automated guidance to an appropriate fingering would be helpful as they learn to play the instrument and develop their own playing technique.

1.1 Purpose

The purpose of this study was to develop a proof of concept that a piano player can use to obtain a suggested fingering. The program takes a musical sequence in MIDI format (from a file or directly from a MIDI keyboard) as input and outputs the suggested fingering as a sequence of digits symbolizing the different fingers. Previous research performed in this field was studied and some of the developed algorithms were implemented in the proof of concept. The difficulties that arise

when trying to develop a working software of this kind are also discussed. The programming of the algorithm was done in Java.

1.2 Problem statement

The goal of this study was to create an algorithm that produces viable piano fingerings for a given musical sequence. The algorithm uses the rules presented by Parncutt et al. (see *Chapter 2, Background* for more information) and is limited to monophonic melodies with interleaved triads. The study examines how articulation can be used to relax the requirements on the algorithm (e.g. whether it is appropriate to reset the hand position after playing a staccato note). Consequently, the objective of this study was to answer the question:

> *"Is it possible to create an algorithm that produces a reasonable piano fingering and add support for interleaved triads and different articulations to the set of rules presented by Parncutt et al.?"*

1.3 Terminology

Fingering - The choice of which fingers and hand positions to use when playing certain musical instruments.
Legato - A form of musical articulation that indicates that musical notes are played or sung smoothly and connected.
Marcato - A musical instruction indicating a note, chord, or passage is to be played louder or more forcefully than surrounding music.
MIDI - Musical Instrument Digital Interface. A technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another.
Monophony - A musical texture that consists of a single melodic line.
Polyphony - A texture consisting of two or more simultaneous lines of independent melody.
Staccato - A form of musical articulation. In modern notation it signifies a note of shortened duration.
Triad - A set of three notes that can be stacked in thirds.
Dynamic programming - A method for solving complex problems by breaking them down into simpler subproblems, eliminating overlapping subproblems.

Chapter 2

Background

Previous research in this field has been rather sparse. One of the more substantial articles on this topic was published by Parncutt et al. (1997) and presents a model that takes a musical sequence of eight notes as input and produces a suggested fingering as output. The model is based on twelve rules that assign degrees of difficulty to a certain finger pair (see Section *3.2* for a complete list of rules). The rules are mainly based on ergonomic constraints of the hand, such as the difficulty to stretch two fingers to reach certain keys and the general weakness of the little finger. All input is interpreted as playing legato and the model ignores the variations in different pianists' hand sizes. To narrow down the number of possible fingerings before assigning difficulties based on the rules, the authors decided to set limits to the maximum and minimum amount of semitones that a certain finger pair can play. Any suggested fingering that contains a finger pair exceeding this span will be ignored.

The experiment was made by letting 28 pianists write down their preferred fingerings of selected musical sequences. These fingerings were then compared to the fingerings proposed by the model. The results are comprehensive and the authors state that "in general, the model successfully predicts the most commonly selected fingerings out of the typically very large number of possible fingerings".

Other research includes a study by Al Kasimi, Nichols and Raphael (2005) which differs from the work of Parncutt et al. in that the authors choose to include chords when calculating the difficulty of a certain fingering. The difficulty is calculated through the use of what the authors call *vertical* and *horizontal* costs. The vertical cost "corresponds to the stretch induced by a given hand position, where the value of the cost is proportional to the difficulty of the stretch." The horizontal cost function on the other hand "accounts for transitions between two successive chords." The authors stress that the research is a work in progress and the credibility of the first preliminary results should therefore be regarded as not too high. The article also lacks information about the methods used to achieve the results.

The study by Hart, Bosch and Tsai (2000) focuses on finding optimal fingerings for monophonic sequences played with the right hand, similar to the previously

mentioned model developed by Parncutt et al. The difficulty of playing two notes in sequence with two different fingers is calculated through the use of four rules. The rules assign a difficulty level based on the color of the keys of the played notes, and the number of seminotes in between. For example, playing two white keys with a certain seminote interval will yield a different difficulty than when the lower-pitched note is black and the higher-pitched key is white. The color of the keys is the only aspect taken into account when calculating the difficulty, compared to Parncutt et al.'s model which implements several other parameters as well.

The report contains a thorough description of how to use dynamic programming to produce an optimal fingering but no test results are presented. The execution of the algorithm is claimed to require "just a fraction of a second of CPU time to finger a 65-interval righthand passage" and the authors' focus seems to be on how the model can be used rather than on how accurate it is.

After examining these three reports it was clear that Parncutt et al.'s model and research was the most comprehensive and sophisticated. They also present a good amount of results and a discussion of how the model can be improved.

Chapter 3

Method

3.1 Algorithm

The algorithm will be described as a recursion. It should, however, be implemented using dynamic programming (dynprog) for efficiency. The idea of the algorithm is to generate all possible fingerings which can be illustrated with an exponentially growing tree of growth rate 5^n (*Figure 3.1*). Each finger is represented by a number; 1 = the thumb, 2 = the index finger, and so forth. The fingerings are represented as paths in the tree (excluding the first node). Some of the fingerings, however, are ergonomically infeasible (such as 2 3 2 3 2 for a one-directional sequence) and can be eliminated prior to knowing the musical sequence by defining which finger transitions are allowed. The difficulty of the fingerings is thereafter calculated by a set of rules which is the core of the algorithm. To minimize the computation load, only paths of the same length as the sequence are processed.

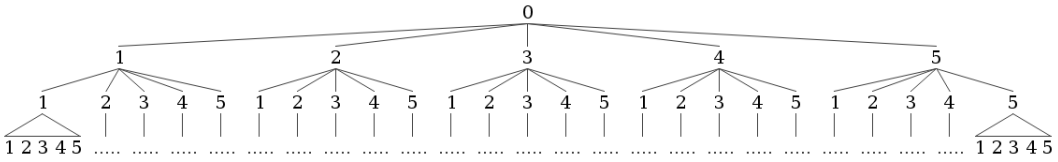

Figure 3.1. The tree of generated fingerings.

The core of the algorithm consists mainly of the twelve rules presented by Parncutt et al. (described in Sections *3.2.1* - *3.2.11*). The rules are based purely on the ergonomic limits of the hand denoted with the stretch spans between finger pairs in semitones - divided in three different states, when the stretch can be accomplished while the hand is relaxed (Rel), when the stretch is somewhat larger but can still be accomplished comfortably (Comf) and when the stretch is even larger but still can be accomplished practically (Prac). These limitations will be referred to simply by their abbreviations noted in the parentheses. The determination of the difficulty of

a specific fingering is based on the sum of the values assigned by all the rules (e.g. the higher the sum - the harder the fingering). The algorithm is also extended by two rules to increase efficiency and usability (sections *3.2.12* - *3.2.13*).

3.2 Rules

The following sections briefly describe the rules used to calculate the difficulty of a specific fingering. All rules were written in Java code but in order to enhance the readability their implementations are presented in pseudo-code. In these pseudo-code snippets each finger is symbolized by a number from 1 to 5; 1 being the thumb and 5 being the little finger.

3.2.1 Stretch

This rule increases the difficulty by a constant whenever the stretch exceeds the bounds of (Conf) for a finger pair.

```
1 //if the stretch is lower than the minimum or higher than the maximum
2 //comfortable span for the first and second finger in
3 //a group of two increase the difficulty by a constant
4 if (handSpan.MinComf(fing1, fing2) > stretch || (handSpan.MaxComf(fing1,
5 fing2) < stretch) {
6         increaseDifficulty(Constant);
7 }
```

3.2.2 Small and Large-span

These two rules increase the difficulty by a value relative to the number of semitones that exceeds the bounds of (Rel) for a finger pair. This value is multiplied by 2 if the finger pair does not include the thumb.

```
1  //if the stretch is lower than the minimum or higher than the maximum
2  //relaxed span for the first and second finger in
3  //a group of two
4  if (handSpan.MinRel(fing1, fing2) > stretch
5         || (handSpan.MaxRel(fing1, fing2) < stretch) {
6
7         //if either the first or second finger is the thumb
8         //increase the difficulty by a relative constant
9         if (fing1 == 1 || fing2 == 1) {
10                increaseDifficulty(Constant * exceededSemiTonesCount);
11        }else{
12                increaseDifficulty(Constant * exceededSemiTonesCount * 2);
13        }
14 }
```

3.2.3 Position-Change-Count

This rule increases the difficulty by a constant whenever the hand changes position. A hand position change is divided in two levels; a full change and a half change. The difference is that full changes require more effort, thus the constant is multiplied by 2. A hand position change occurs whenever the bounds of (Comf) are exceeded by the first and third fingers that play a group of three notes. A full change occurs when the second finger is the thumb; the pitches of the three notes go in the same direction (either rising or falling) and not exceeding the bounds of (Prac).

```
1  //if the stretch is lower than the minimum or higher than the maximum
2  //comfortable span for the first and second finger in
3  //a group of two
4  if (handSpan.MinComf(fing1, fing3) > stretch || (handSpan.MaxComf(fing1,
       fing3) < stretch) {
5
6          //if the second finger is the thumb
7          //and the notes go in the same direction (pitch wise)
8          //and the span of the first and second note is lower or equal to
9          //the the maximum practical span for the first and third fingers
10         if(fing2 == 1 && sameDirection(note1, note2, note3) &&
               noteSpan.getSpan(note1, note3) <= handSpan.MaxPrac(fing1,
               fing3)){
11                 increaseDifficulty(Constant * 2);
12         }else{
13                 increaseDifficulty(Constant);
14         }
15 }
```

3.2.4 Position-Change-Size

This rule increases the difficulty by a value relative to the number of semitones that exceeds the bounds of (Comf) for the first and third notes in a group of three.

```
1  //if the stretch is lower than the minimum or higher than the maximum
2  //comfortable span for the first and third finger in
3  //a group of three
4  if (handSpan.MinComf(fing1, fing3) > stretch || (handSpan.MaxComf(fing1,
       fing3) < stretch) {
5          increaseDifficulty(Constant * exceededSemiTonesCount);
6  }
```

3.2.5 Weak-Finger

This rule increases the difficulty by a constant whenever the fourth or fifth finger are used, assuming that they are weaker than the other fingers.

```
//if finger 4 or 5 is used increase difficulty by a constant
if(fing1 == 4 || fing1 == 5){
        increaseDifficulty(Constant);
}
```

3.2.6 Three-Four-Five

This rule increases the difficult by a constant whenever the third, fourth and fifth finger are used consecutively, in any permutation.

```
//if any permutation of fingers 3, 4, 5 is used in a group
//of three, increase the difficulty by a constant
if(sort(fing1, fing2, fing3) == {3, 4, 5}){
        increaseDifficulty(Constant);
}
```

3.2.7 Three-to-Four

This rule increases the difficulty by a constant whenever the third finger is immediately followed by the fourth finger.

```
//if finger 3 is followed by finger 4,
//increase the difficulty by a constant
if(fing1 == 3 && fing2 == 4)){
        increaseDifficulty(Constant);
}
```

3.2.8 Four-on-Black

This rule increases the difficulty by a constant whenever the third and fourth fingers is used consecutively; the third on a white key and the fourth on a black key.

```
//if finger 3 and 4 is used consecutively in any permutation,
//increase the difficulty by a constant
if(sort(fing1, fing2) == {3, 4} && !isOnBlack(fing1) && isOnBlack(fing2)){
        increaseDifficulty(Constant);
}
```

3.2.9 Thumb-on-Black

This rule increases the difficulty by a constant whenever the thumb plays a black key. The constant is increased if the keys played immediately before and after are white.

```
//Increase whenever thumb is on black, increase more if
//preceding and/or the finger immediately after is on white.
if(fing2 == 1 && isOnBlack(fing2)){
        increaseDifficulty(Constant);
        if(!isOnBlack(fing1)){
                increaseDifficulty(Constant);
        }
        if(!isOnBlack(fing3)){
                increaseDifficulty(Constant);
        }
}
```

3.2.10 Five-on-Black

This rule increases the difficulty by a constant whenever the fifth finger plays a black key and the keys played before and/or after are white.

```
if(fing2 == 5 && isOnBlack(fing2)){
        if(!isOnBlack(fing1)){
                increaseDifficulty(Constant);
        }
        if(!isOnBlack(fing3)){
                increaseDifficulty(Constant);
        }
}
```

3.2.11 Thumb-passing

This rule increases the difficulty by a constant whenever the thumb is passed under (or gets passed by) another finger. It increases further if the thumb plays a black note and the other finger plays a white key.

```
//If either of the fingers is the thumb, and it is passing - or
//gets passed by - another finger
if(thumbIsPassedByFinger || fingerIsPassedByThumb){
        increaseDifficulty(Constant);

        //If the thumb is on black and the other finger is on white.
        if(fing1 == 1 && isOnBlack(fing1) && !isOnBlack(fing2)){
                increaseDifficulty(Constant);
        }else if(fing2 == 1 && isOnBlack(fing2) && !isOnBlack(fing1)){
                increaseDifficulty(Constant);
        }
}
```

3.2.12 Sub-Phrase

This rule and the following extends the algorithm developed by Parncutt et al. The sub-phrase rule will reset the algorithm whenever a hand position change is possible without affecting the musical phrase. This kind of behavior happens for example when a rest is inserted or after staccato notes.

```
//if the musical phrase inserted contains any sub-phrasing indicators,
//divide it into smaller sub-phrases and process them separately.
Sequence [] subSeq = seq.split("sub-phrasing-indicator");
for(Sequence s : subSeq){
        generateFingering(s);
}
```

3.2.13 Polyphonic-Melodies

This rule generates fingerings for polyphonic melodies (chords) by treating the input as if it is monophonic where finger passing is not allowed and the span between all pairs of fingers used does not exceed the bounds of (Rel).

```
//text-pseudo example of how a polyphonic melody is identified
//and allowed
isPolyphonic(fing1,...,fingN,note1,...,noteN){
        if(!fingerIsPassedByAnotherFinger){
                for all pairs of fingers:
                                stretch = (fing1, fing2).getStretch(){
                        if(stretch < handSpan.MinRel(fing1, fing2) &&
                                stretch > handSpan.MaxRel(fing1, fing2)){
                                        setImpossible();
                                        return;
                        }else
                                setPossible();
                }
        }else{
                setImpossible();
                return;
        }
}
```

3.3 Evaluation

In order to evaluate the algorithm, a collaboration was initiated with an experienced jazz pianist. The pianist played a monophonic sequence of varying length from three musical pieces; *All of Me* (Marks and Simons 1931), *Ain't Misbehavin'* (Waller, Brooks and Razaf 1929) and *Alla fåglar kommit ren* (children's song). His

suggested fingerings for each sequence were compared with the results generated by the algorithm for the same sequences.

Since all of the sequences were monophonic, an additional musical piece (*Interleaved Triad*) was composed by the authors containing an interleaved triad. This piece would be used to examine the effect of the *Polyphonic-Melodies* rule. In order to evaluate the *Sub-Phrase* rule the first 27 notes of *Für Elise* (Beethoven 1810) were run by the algorithm. This sequence of notes contain rests that would trigger this rule. The fingerings generated by the algorithm for *Interleaved Triad* and *Für Elise* were then evaluated by the authors.

Chapter 4

Results

Figures 4.1 through 4.3 show the musical sequences for (1)*All of Me*, (2)*Ain't Misbehavin'* and (3)*Alla fåglar kommit ren* with the pianist's fingering suggestions above the musical staff (circled). The corresponding least difficult fingerings generated by the algorithm are indicated below the staff.

The figures provide an overview of the notes in the sequence. However, they do not display the actual note duration. The reason for disregarding the duration is because the sequences received from the pianist did not contain this information. Furthermore, there are no rules affected by the duration of a note.

Figures 4.4 and 4.5 display only the fingerings generated by the algorithm.

The pianist's fingering for sequence (1) was found to be the 12th easiest fingering generated by the algorithm. The number of different notes compared to the algorithm's fingering was 4 out of 9 possible.

For sequence (2), the pianist's fingering rank was 15 and the different notes were 5 out of 13.

For sequence (3), the fingerings differed in 11 out of 14 notes and the pianist's fingering was not found by the algorithm. Notice in Figure 4.3 that the middle finger passes over the index finger when going from the 8th to the 9th note. A summary of the ranks and different notes can be found in Table 4.1.

Table 4.1. Number of different notes and pianist fingering rank

Sequence	Notes where finger differ	Total notes	Pianist rank
All of Me	4	9	12
Ain't Misbehavin'	5	13	15
Alla Fåglar Kommit Ren	11	14	-

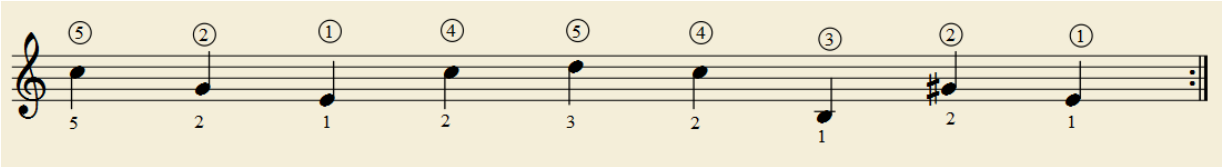

Figure 4.1. Fingerings for *All Of Me*

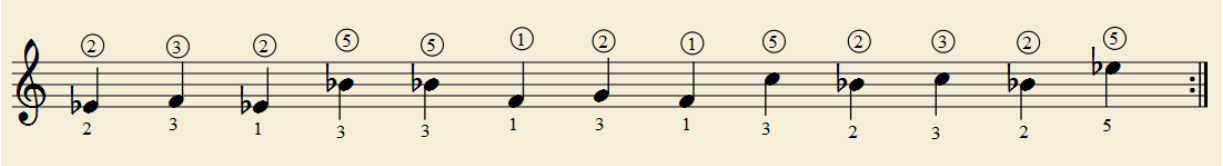

Figure 4.2. Fingerings for *Ain't Misbehavin'*

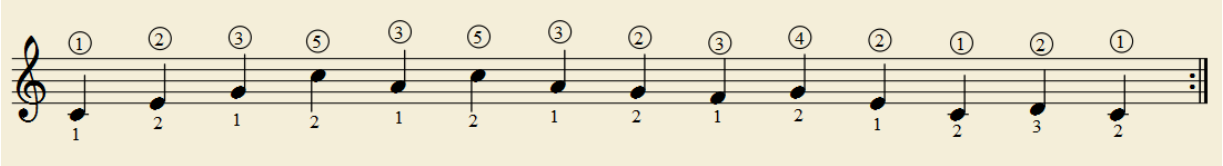

Figure 4.3. Fingerings for *Alla fåglar kommit ren*

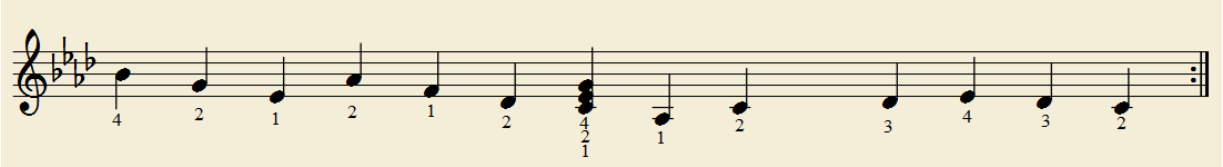

Figure 4.4. Fingerings for *Interleaved Triad*

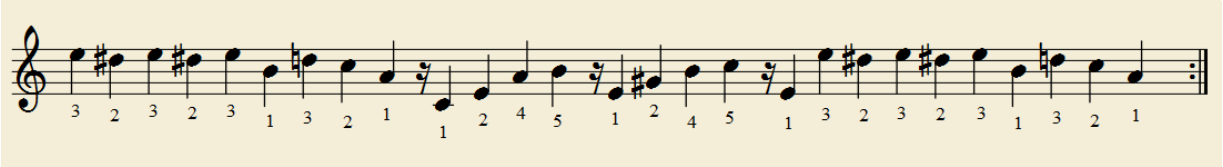

Figure 4.5. Fingerings for *Für Elise*

Chapter 5

Discussion

The difference between the pianist's suggested fingering and the fingering generated by our algorithm for the *All of Me* sequence is likely due to the rules involving the 3rd, 4th and 5th fingers. Especially the Weak-Finger and Three-Four-Five rules influence this difference, since they increase the difficulty by higher values than the stretch rules. That favors the stretch of stronger fingers instead of the use of weaker fingers.

A pianist should, or at least try, to use all fingers. Even weak fingers should be used, that is how finger strength and accuracy is built up. The Weak-Finger rule disregards the fact that an experienced pianist may have significant strength in the 3rd and 4th fingers, thus rendering this rule useless when trying to calculate the best fingering.

The pianist's suggested fingering for the *Ain't Misbehavin'* sequence does not differ much from the one generated by the algorithm. The difference is mainly due to the Five-On-Black and Weak-Fingers rules, as shown at the 4th and 5th notes and the longer stretches between stronger fingers.

The fingering suggested by the pianist for the *Alla fåglar kommit ren* sequence is completely discarded by our algorithm, mainly because of the finger passing between the 8th and 9th notes. The algorithm does not allow finger passings not involving the 1st finger (thumb). This technique is practiced by a minority of pianists and is not advisable. Thus we think that the algorithm did good aside from the exclusive use of the strong fingers, which again is caused mainly by the Weak-Finger rule.

The fingerings generated by the algorithm for the three sequences above is considered to be reasonable aside from being different from the pianist's suggestion.

The fingering generated by the algorithm for the *Interleaved Triad* sequence should be considered a step forward towards generating polyphonic fingerings. The presented fingering is not optimal because this is an experimental version and attention was not paid to all the parameters which should prevent finger intersection. That is, fingers playing notes before and after a chord intersect with fingers used to play the chord. This is shown by the notes immediately before and after the chord in the *Interleaved Triad* sequence.

The fingering generated for the *Für Elise* sequence is considered to be very reliable as there is no extensive stretching of the fingers nor finger passings. This sequence contains rests which triggers the Sub-Phrase rule. The Sub-Phrase rule divides the sequence into smaller sequences which eliminates finger stretches. Aside from that, it improves the runtime of the algorithm considerably which makes processing of longer sequences, like *Für Elise*, possible.

One problem found when testing the proof of concept was that it was extremely difficult to determine the relevance of some rules, due to the large amount of rules that Parncutt et al. use. When trying to optimize the algorithm to produce fingerings that resembled the pianist's fingerings, we experimented with disregarding the rules that increased the difficulty when using the third, fourth and fifth finger. The results showed that for some musical sequences the suggested fingerings came closer to the pianist's fingerings, whereas for other sequences the fingerings differed even more. This suggests that some of the rules may be superfluous for certain musical sequences. Examining what type of sequences require - or do not require - the use of certain rules is beyond the scope of this study. Such an analysis is, however, necessary in order to improve the performance of the algorithm significantly.

The main problem with designing a software intended to help pianists is the large amount of parameters that needs to be taken into account during the implementation process. Hand size, finger strength and stretchability are just a few of the parameters that differ from person to person and that need to be carefully implemented in order to provide an efficient model for producing a keyboard fingering.

Chapter 6

Conclusion

The proposed algorithm was not able to suggest any fingerings that were similar to the ones provided by the experienced pianist for the monophonic sequences. However, the Sub-Phrase rule made the algorithm significantly faster while still providing a reasonable fingering. The Polyphonic-Melodies rule contributed to creating a fingering that was not possible to play, but a few adjustments would probably make it useful. More sequences containing interleaved triads and sub-phrases would need to be analyzed before concluding the consistency and power of the added rules.

In conclusion, it was possible to add support for interleaved triads and different articulations to the set of rules presented by Parncutt et al. and create an algorithm that produces reasonable piano fingerings.

Bibliography

[1] Parncutt R., Sloboda J., Clarke E., Raekallio M. and Desain P. (1997). An ergonomic model of keyboard fingering for melodic fragments. *Music Perception*, Vol.14, p. 341-382 .

[2] Kasimi A., Nichols E. and Raphael C. (2005). Automatic fingering system (AFS). Proc. 6th International Conference on Music Information Retrieval ISMIR.

[3] Hart M., Bosch R. and Tsai E.,(2000). Finding optimal piano fingerings. *The Undergraduate Mathematics and Its Applications Journal*, UMAP, p. 1-10.

[4] Marks G. and Simons (1931). All of Me.

[5] Waller F., Brooks H. and Razaf A. (1929). Ain't Misbehavin'.

[6] Beethoven L. (1810). Bagatelle No. 25 in A minor (WoO 59) (Für Elise).

[7] "Fingering." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 4 April 2013. Web. 12 April 2013. <http://en.wikipedia.org/wiki/Fingering>

[8] "Legato." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 12 March 2013. Web. 12 April 2013. <http://en.wikipedia.org/wiki/Legato>

[9] "Marcato." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 6 April 2013. Web. 12 April 2013.<http://en.wikipedia.org/wiki/Marcato>

[10] "MIDI." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 11 April 2013. Web. 12 April 2013.<http://en.wikipedia.org/wiki/Midi>

[11] "Monophony." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 11 April 2013. Web. 12 April 2013.<http://en.wikipedia.org/wiki/Monophony>

[12] "Polyphony." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 21 Feb. 2013. Web. 12 April 2013.<http://en.wikipedia.org/wiki/Polyphony>

[13] "Staccato." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 7 April 2013. Web. 12 April 2013.<http://en.wikipedia.org/wiki/Staccato>

[14] "Triad." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 11 April 2013. Web. 12 April 2013.<http://en.wikipedia.org/wiki/Triad_(music)>

[15] "Dynamic programming." *Wikipedia: The Free Encyclopedia.* Wikimedia Foundation, Inc. 9 April 2013. Web. 12 April 2013.<http://en.wikipedia.org/wiki/Dynamic_programming>