Generating fingerings for polyphonic piano music with a tabu search algorithm

Generating Fingerings for Polyphonic Piano Music with a Tabu Search Algorithm

Matteo Balliauw, Dorien Herremans,
Daniel Palhazi Cuervo, and Kenneth Sörensen

University of Antwerp, Faculty of Applied Economics,
Prinsstraat 13, 2000 Antwerp, Belgium
{matteo.balliauw,dorien.herremans,daniel.palhazicuervo,
kenneth.sorensen}@uantwerpen.be
http://www.uantwerpen.be

Abstract. A piano fingering is an indication of which finger is to be used to play each note in a piano composition. Good piano fingerings enable pianists to study, remember and play pieces in an optimal way. In this paper, we propose a tabu search algorithm to find a good piano fingering automatically and in a short amount of time. An innovative feature of the proposed algorithm is that it implements an objective function that takes into account the characteristics of the pianist's hand and that it can be used for complex polyphonic music.

Keywords: Piano Fingering, Tabu Search, Metaheuristics, OR in Music, Combinatorial Optimisation

1 Introduction

Both pianists and composers often add a fingering to sheet music in order to indicate the appropriate finger that should be used to play each note. *Piano fingerings* are important as not every finger is equally suited to play each note, and some combinations of fingers are better suited to play certain note sequences than others [**?**]. Additionally, having a well thought-out piano fingering can help a pianist to study and remember a piece. It can also enhance the interpretation and the musicality of a performance [**?**].

Developing a high-quality fingering requires a considerable amount of time and expertise. For that reason, in this paper we develop an algorithm to automatically determine a good piano fingering. The purpose of the algorithm is to help pianists with little experience in deciding on a good fingering, as well as all pianists who quickly want to obtain a fingering they can use as a starting point.

In this paper, we model the generation of a good piano fingering as a combinatorial optimisation problem, in which a finger has to be assigned to every note in the piece. The quality of a fingering is evaluated by means of an objective function that measures playability. This objective function takes into account the characteristics of the player's hands. The proposed algorithm is a tabu search (TS) heuristic, capable to find a good fingering solution for a complex polyphonic

piano piece in a reasonable amount of execution time. This technique has been successfully used in the field of computer-aided composing [**?**].

This paper is divided into six sections. Section **??** gives a short overview of the literature concerning the generation of piano fingerings and algorithms developed to solve this problem. In Section **??**, a mathematical formulation of the problem is introduced. Section **??** explains the tabu search algorithm in detail. In Section **??**, we show and discuss an example of a fingering generated by the TS for an existing piece of music. The final section gives some conclusions and suggestions for future research.

2 Literature Review

In the 18th century, exercises to learn frequently used fingering combinations fluently were published for the first time. Carl Philipp Emanuel Bach was among the first musicians to write down an entire set of rules for piano fingering. Previously, such rules were only transmitted orally through lessons. In the 19th century, many composers followed his example [**?**]. At the end of the 20th century, the first attempts to generate a mathematical formulation that could model a piano fingering and develop a suitable algorithm were published, as we show in the next subsection. Similar research has been carried out among others for string instruments [**?**].

2.1 Fingering Quality

In order to evaluate a piano fingering, it is necessary to quantify it quality. Three important dimensions of a piano fingering add to this quality. The main element is the ease of playing and this is where this research focuses on. Additional dimensions that add to the quality of a fingering are the ease of memorisation and the facilitation of the interpretation [**?**].

The dominant approach for evaluating the quality of a piano fingering using an objective function, which is also followed in this research, is based on hand-made rules [**?**][**?**][**?**]. Other strategies include using a machine learning approach such as Markov Models based on transition matrices [**?**][**?**][**?**][**?**] or Hidden Markov Models (HMM) [**?**].

In order to evaluate the playability of a fingering, the set of rules proposed by Parncutt et al. [**?**] and Parncutt [**?**] serve as the basis for this research. Every source of difficulty in a fingering, both monophonic and polyphonic, is assigned a cost. Every cost factor is attributed a weight. The playability measure is an objective function that consists of the weighted sum of costs and is to be minimised. This approach has the advantage that pianists can express their trade-offs between the sources of difficulty. This can be done by assigning different weights to each source of difficulty in the objective function. Parncutt's original cost factors for piano fingering have been expanded by Jacobs [**?**] and Lin and Liu [**?**]. More recently, some improvements and additions were made by the authors of this paper [**?**]. This latter set of rules is used in this research. Issues such as personal

preferences, the use of the left hand and the difference between playing a black and white key have been ignored in the past [**?**], but are taken into account in this paper.

2.2 Algorithms for Automatic Generation of Fingerings

In many papers, dynamic programming is used to find optimal piano fingerings according to the selected objective function. Dijkstra's algorithm is used by Parncutt et al. [**?**] on monophonic music and by Al Kasimi et al. [**?**] on monophonic pieces with some simple polyphonic chords. Similar dynamic programming algorithms have been used in literature by Robine [**?**] and Hart et al. [**?**] on monophonic music. A rule-based expert system that optimises a fitness function for similar music was developed by Viana et al. [**?**] to implement their Intelligent System for Piano Fingering Learning Aid (SIEDP). For the HMM-model in monophonic music, Yonebayashi et al. used a Viterbi algorithm [**?**]. Although some improvements have been made to reduce the computing time of the aforementioned algorithms, it is often impossible for them to deal with complex polyphonic pieces (where simultaneous notes can have different starting and ending times and where hence a graphical representation would no longer work). For this reason, we develop an algorithm that can deal with complex polyphony in an effective and efficient way. This algorithm is explained in Section **??**. In the next section, we mathematically formulate the problem of finding a piano fingering .

3 Problem Description

When generating a piano fingering, it is necessary to decide which finger should play each note. Each finger is represented using the traditional coding from 1 (thumb) to 5 (little finger).

In order to consider piano fingering as an optimisation problem, we define an objective function that measures the quality of a solution by looking at the playability of the piece. This objective function was described by Balliauw [**?**] as an adaptation of the work of Parncutt et al. [**?**] and Jacobs [**?**] and allows to work with both the right and the left hand.

The objective function takes into account a distance matrix, displayed in Table **??**. This contains information for each finger pair about the distances that are easy and difficult to play, respectively called `Rel` (relaxed range), `Comf` (comfortable range) and `Prac` (practically playable range). These allowed distances can be adapted by the user of the algorithm according to the biomechanics of his or her hand.

The objective function consists of three sets of rules. Using the distance matrix, a first set of rules compares the actual distance (calculated by subtracting the corresponding values of the keys in Fig. **??**) and the allowed distance between two simultaneous or consecutive notes for the proposed pair of fingers to play these two notes. A penalty score is applied when the actual distance is larger

than the different types of allowed distances. As Jacobs [**?**] argued, the distances in the previous research of Parncutt et al. [**?**] were not accurate. To calculate these penalties more accurately, the authors increased in previous work [**?**] the distances between E and F and between B and C to two half notes, to equal the distances between other adjacent white keys on a piano keyboard, as illustrated in Fig. **??**. The second set of rules is implemented to prevent unnecessary and inconvenient hand changes. The third group of rules prevents difficult finger movements in monophonic music. In this third set, two additional rules proposed by Balliauw [**?**] promote the choice of logical, commonly used fingering patterns. For a more detailed discussion of the rules and distances, we refer to the actual references.

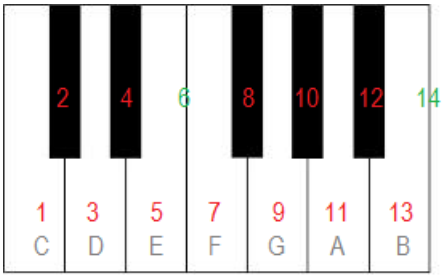


Fig. 1. Piano keyboard with additional imaginary black keys [**?**].

The weighted penalty scores of all rules together, displayed in Table **??**, are summed to obtain the objective function value. As this value indicates the difficulty of the fingering, it has to be minimised.

Table 1. Example distance matrix that describes the pianist's right hand [**?**].

Finger pair	`MinPrac`	`MinComf`	`MinRel`	`MaxRel`	`MaxComf`	`MaxPrac`
1-2	-10	-8	1	6	9	11
1-3	-8	-6	3	9	13	15
1-4	-6	-4	5	11	14	16
1-5	-2	0	7	12	16	18
2-3	1	1	1	2	5	7
2-4	1	1	3	4	6	8
2-5	2	2	5	6	10	12
3-4	1	1	1	2	2	4
3-5	1	1	3	4	6	8
4-5	1	1	1	2	4	6

Table 2. Set of rules composing the objective function [**?**].

Rule	Application	Description	Score
1	All	For every unit the distance between two consecutive notes is below `MinComf` or exceeds `MaxComf`.	+2
2	All	For every unit the distance between two consecutive notes is below `MinRel` or exceeds `MaxRel`.	+1
3	Monophonic	If the distance between a first and third note is below `MinComf` or exceeds `MaxComf`: add one point. In addition, if the pitch of the second note is the middle one, is played by the thumb and the distance between the first and third note is below `MinPrac` or exceeds `MaxPrac`: add another point. Finally, if the first and third note have the same pitch, but are played by a different finger: add another point.	+1 +1 +1
4	Monophonic	For every unit the distance between a first and third note is below `MinComf` or exceeds `MaxComf`.	+1
5	Monophonic	For every use of the fourth finger.	+1
6	Monophonic	For the use of the third and the fourth finger (in any consecutive order).	+1
7	Monophonic	For the use of the third finger on a white key and the fourth finger on a black key (in any consecutive order).	+1
8	Monophonic	When the thumb plays a black key: add a half point. Add one more point for a different finger used on a white key just before and one extra for one just after the thumb.	+0.5 +1 +1
9	Monophonic	When the fifth finger plays a black key: add zero points. Add one more point for a different finger used on a white key just before and one extra for one just after the fifth finger.	+1 +1
10	Monophonic	For a thumb crossing on the same level (white-white or black-black).	+1
11	Monophonic	For a thumb on a black key crossed by a different finger on a white key.	+2
12	Monophonic	For a different first and third note, played by the same finger, and the second pitch being the middle one.	+1
13	All	For every unit the distance between two following notes is below `MinPrac` or exceeds `MaxPrac`.	+10
14	Polyphonic	Apply rules 1, 2 (both with doubled scores) and 13 within one chord.	
15	All	For consecutive slices containing exactly the same notes (with identical pitches), played by a different finger, for each different finger.	+1

The problem description also takes into account one hard constraint. This constraint enforces that a single finger cannot be used to play two different, simultaneous notes, as this is not feasible to execute on a real piano.

4 Tabu Search Algorithm

A common approach to solve combinatorial optimisation problems is the use of exact algorithms, that ensure finding the optimal solution. The problem with this approach is that in the worst case the execution time can grow exponentially with the size of the instance treated. For this reason, they are often suited to deal with short or simple instances only. Metaheuristics form an alternative approach to generate good solutions for complex problems (often involving large and complex instances) in a reasonable amount of execution time. Several metaheuristic frameworks have been proposed in the literature, all of which offer guidelines to build heuristic algorithms [**?**].

Different categories of metaheuristics exist, such as constructive, population-based and local search [**?**]. In this research we develop a local search heuristic to generate piano fingerings for complex polyphonic music, as this class of heuristics better allows to take the characteristics of the problem into account [**?**]. Local search heuristics start from a *current solution*, to which small, incremental changes are made, called *moves*. All moves performing the same changes are part of the same *move type*. The set of solutions that can be reached from the current solution by a certain move type is called the *neighbourhood* of the solution. A *local optimum* is reached when the neighbourhood contains no improvement for the current solution [**?**].

An algorithm can escape from such a local optimum or avoid getting trapped into cycles by using a *tabu list*. This list contains a number of moves that were performed right before the current move and that are excluded from the neighbourhood of the current solution. These moves are *tabu active*. The length of the tabu list is called *tabu tenure*. The move with the best objective function value from this neighbourhood is performed, even if it worsens the current solution. In this way, the algorithm avoids getting trapped into cycles and can explore the solution space around the local optimum to eventually arrive at a better solution. The tabu tenure and the number of allowed iterations without improvement are the two parameters that define a *tabu search* [**?**].

When no more improvements can be made to a local optimum within one neighbourhood, the algorithm switches to another neighbourhood, defined by a different move type. This strategy is similar to that implemented by the *Variable Neighbourhood Search (VNS)* [**?**].

In this paper, finding a good piano fingering is modelled as a combinatorial optimisation problem. When this problem becomes larger (as is often the case), the number of possible solutions grows exponentially with the number of notes in the piece (5^n). To this end, we chose to develop a tabu search algorithm (TS), as it has already been applied efficiently to many other combinatorial optimisation problems such as the NP-hard travelling salesman problem [**?**] and

vehicle routing problem [**?**]. More recently, it has also been used in the field of music [**?**]. As a result, developing a TS algorithm was considered as a viable choice in this paper.

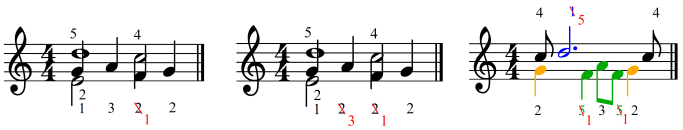
The TS algorithm, displayed schematically in Algorithm **??**, starts from a random initial solution, and optimises the fingering for the right hand. The optimisation processes for the left hand is identical and executed subsequently. First, a preprocessing step Swap is applied to the initial solution. This step makes significant improvements (i.e., reducing the objective function value $f(\mathcal{S})$) by swapping two fingers throughout the entire piece. This preprocessing step has a positive impact on the solution quality and reduces the required execution time.

Algorithm 1: TS algorithm for each hand

Input : File $\mathcal{F}$ containing the piece
Output: File $\mathcal{F}'$ with the generated fingering included

$\mathcal{P} \leftarrow$ Parse($\mathcal{F}$)
$\mathcal{P} \leftarrow \mathcal{P}/\{$non-used hand$\}$
$\mathcal{S}_{best} \leftarrow$ Rand_Sol($\mathcal{P}$)
$\mathcal{S}_{best} \leftarrow$ Swap($\mathcal{S}_{cur}$)
`tabutenure` $= 0.5 \cdot$Number_Notes($\mathcal{P}$)
`maxiters` $= 5 \cdot$`tabutenure`
Init_Tabu_List(`tabutenure`)
$\mathcal{T} \leftarrow$ True
while $\mathcal{T} =$ *True* **do**
  $\mathcal{T} \leftarrow$ False
  for $o \in \mathcal{O}$ **do**
    $\mathcal{S}_{nbh} \leftarrow \mathcal{S}_{best}$
    $\mathcal{S}_{cur} \leftarrow \mathcal{S}_{best}$
    $i = 0$
    while $i <$ *maxiters* **do**
      $\mathcal{N} \leftarrow$ Neighbourhood$_o(\mathcal{S}_{cur})$
      $\mathcal{N} \leftarrow \mathcal{N}/\{s : s$ involve $t_{ij} \in$ Tabu_List$\}$
      $\mathcal{S}_{cur} \leftarrow$ Best_Sol($\mathcal{N}$)
      Update_Tabu_List()
      if $f(\mathcal{S}_{cur}) < f(\mathcal{S}_{nbh})$ **then**
        $\mathcal{S}_{nbh} \leftarrow \mathcal{S}_{cur}$
        $i = 0$
      else
        $i \leftarrow i + 1$
    if $f(\mathcal{S}_{nbh}) < f(\mathcal{S}_{best})$ **then**
      $\mathcal{S}_{best} \leftarrow \mathcal{S}_{nbh}$
      $\mathcal{T} \leftarrow$ True
    Clear_Tabu_List()
$\mathcal{F}' \leftarrow$ Write($\mathcal{S}_{best}$)

The main loop of the algorithm uses three neighbourhood operators, each defined by a move type and executed consecutively. For each operator, the best solution from the neighbourhood is selected with a steepest descent strategy. As a result, the move that leads to the best fingering is chosen at each iteration. A first neighbourhood, called Change1, is defined by a move type that changes the finger of a note to any other possible finger. The move type Change2 is similar to Change1, but it is expanded to two adjacent or simultaneous notes. This move is useful in situations in which changing two adjacent notes simultaneously improves the solution but changing each note separately has a detrimental effect on the fingering and is therefore discarded by Change1. To increase the interchangeability of two fingers in polyphonic music, moves of the third type SwapPart change the fingering of a note a from finger k to l, where the fingering of all notes b (played with finger l, starting before the end of note a and ending after the start of note a) are changed from l to k. An illustrative example of each neighbourhood operator $o \in \mathcal{O} = \{\text{Change1}, \text{Change2}, \text{SwapPart}\}$ can be found in Fig. **??**.



(a) Change1 (b) Change2 (c) SwapPart

Fig. 2. Examples of each move explored in every neighbourhood considered by the tabu search algorithm.

To perform tabu search using a neighbourhood, a *tabu list* is initialized after the preprocessing step. The parameter `tabutenure` is defined as a percentage of the number of notes played by a given hand. The tabu list considers changing or swapping the fingering of a specific note i to fingering j as a forbidden move if the couple t_{ij} is *tabu active*, i.e., it is on the tabu list. A couple t_{ij} becomes tabu active after the fingering of note i has been moved to finger j and remains active for a number of moves, equal to the tabu tenure. As a result, it is possible that moving to the best neighbouring solution (that is not tabu) reduces the quality of the current solution. The number of allowed iterations without improvement is defined as a parameter, `maxiters`. This enables the algorithm to explore the solution space around the current solution and escape from a local optimum, given that enough iterations without improvement are allowed (here defined as a percentage of the tabu tenure). A path to arrive in a different area of the solution space and escape the local optimum can thus be pursued. This can be observed in Fig. **??**.

When the number of non-improving iterations using a certain neighbourhood reaches `maxiters`, the content of the tabu list is cleared and in order to escape from the local optimum, the algorithm switches to the next neighbour-

hood. When a loop through all neighbourhoods $o \in \mathcal{O}$ successfully improves the solution $\mathcal{S}$, a new iteration over all these neighbourhoods is executed. Otherwise, the stopping criterion is met and the algorithm returns the best solution found. This solution is outputted to a MusicXML file, which can be processed by open source music sheet software, like MuseScore[1].

5 Results

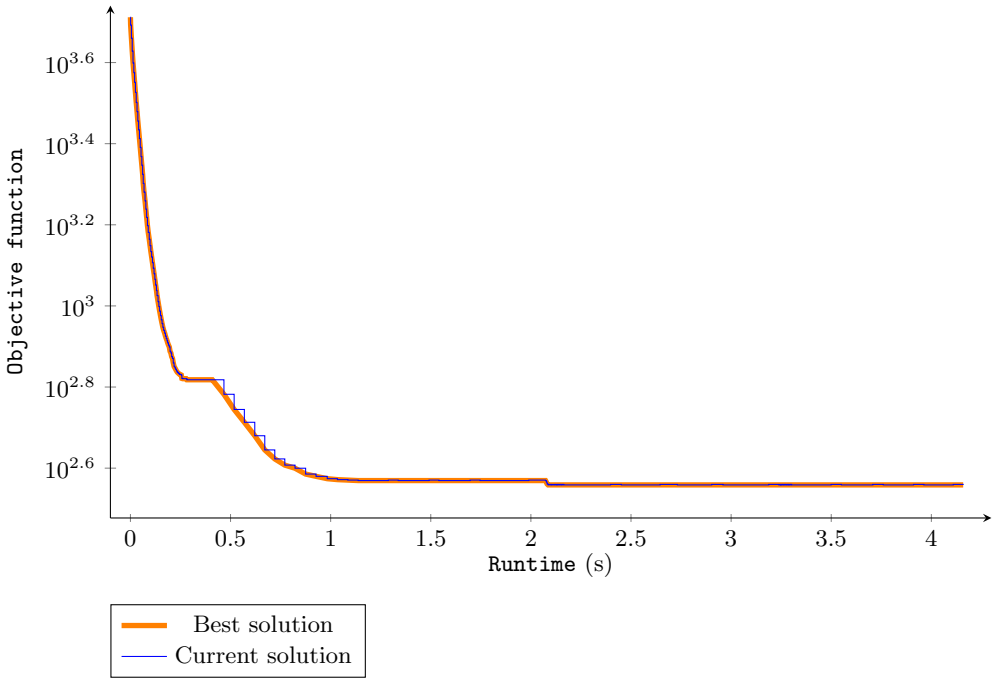


Fig. 3. Evolution of the right hand score over time in the first variation on the Saraband from Suite in D minor (HWV 437) by G.F. Händel.

Fig. **??** shows an example output of the described TS for the first variation on the Saraband from G.F. Händels Suite in D minor (HWV 437). The algorithm was run with all neighbourhood operators $o \in \mathcal{O}$ activated, the tabu tenure set as 50% of the number of notes in the hand and the allowed iterations without improvement set as 5 times the tabu tenure (i.e., 500%). These parameter settings were chosen during a limited pilot study. The fingerings were generated using the hand data shown in Table **??** and with the weights of the rules in the objective function set to 1.

[1] Available from musescore.org

The output of the algorithm shows the first eight bars of the piece. The execution time is very short (5 seconds in total and 4 seconds for the right hand) and the objective function value was 712 in total (362 for the right hand). The evolution of the right hand score over time is displayed in Fig. **??**. Experts (researchers and pianists) confirmed that the solution displayed in the output in Fig. **??** is easily playable and thus forms a good fingering. However it is not perfect, as for example can be seen in the fifth bar where the first note (A) is played with finger 1, and the second (G), lower note with a 4. The algorithm might be improved to reach even better solutions in future research.

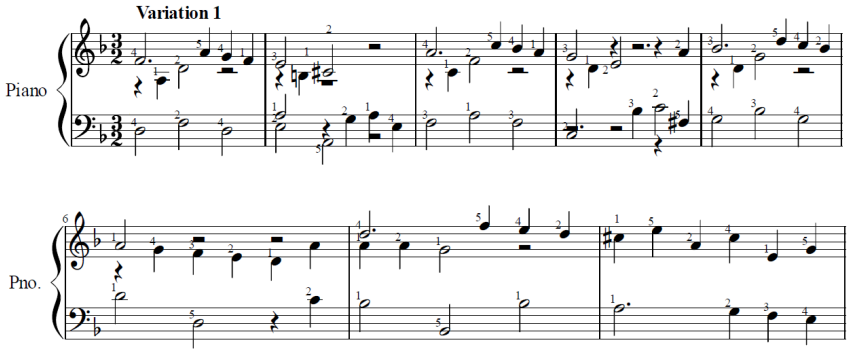


Fig. 4. Output for the first eight bars of the first variation on the Saraband from Suite in D minor (HWV 437) by G.F. Händel.

6 Conclusion and Future Research

In this paper, we described the problem of finding a good piano fingering as a combinatorial optimisation problem. Different sources of difficulty modelled in the implemented objective function allow to deal with complex polyphony, to analyse the left and right hand, and to have the option to adapt some parameters to personal preferences and biomechanics of the hand. We proposed a tabu search algorithm for the generation of piano fingerings, minimising the objective function value. By generating a piano fingering for an existing piece, we showed that the algorithm can find a good solution in a relatively short amount of execution time.

In the future, a possible enhancement of the objective function could be an even more accurate or detailed keyboard distance definition, accounting for the asymmetrical positioning of the black keys. The algorithm could also be expanded by including rules in the objective function that specify the interpretation and memorisation aspects of a piano fingering. Examples of such rules

could be the use of a strong finger (e.g., the thumb) on the first beat of a bar, and using the same fingering patterns for identical note sequences. The objective function could also be improved by integrating models built by applying machine learning techniques to a database of existing piano fingerings. In this way, new rules could be defined and the weights of the different penalty scores in the objective function could be optimised. Further adaptations of the musical rules and interpretation of the outputs should always be verified by existing sheet music and expert piano professionals. The algorithm could also benefit from the inclusion of extra, more sophisticated neighbourhoods.

By taking into account musical sentences, a further improvement in both solution quality and execution time could be attained. The algorithm now improves an entire piece at once. When the piece would be split up into smaller parts based on musical sentences, the decreased size of the neighbourhoods would significantly speed up the execution time of the algorithm. Rules based on these musical sentences might equally improve the solution quality.

Acknowledgments. This research is supported by the Interuniversity Attraction Poles (IAP) Programme initiated by the Belgian Science Policy Office (COMEX project).

References

1. Sloboda, J.A., Clarke, E.F., Parncutt, R., Raekallio, M.: Determinants of Finger Choice in Piano Sight-Reading. J. Exp. Psychol.-Hum. Percept. Perform. 24, 185–203 (1998)
2. Robine, M.: Analyse Automatique du Doigté au Piano. In: Proceedings of the Journées d'Informatique Musicale, pp. 106–112 (2009)
3. Herremans, D.: Tabu Search voor de Optimalisatie van Muzikale Fragmenten. Master's thesis at University of Antwerp, Faculty of Applied Economics, Antwerp (2005)
4. Gellrich, M., Parncutt, R.: Piano Technique and Fingering in the Eighteenth and Nineteenth Centuries: Bringing a Forgotten Method Back to Life. B. J. Music Ed. 15, 5–23 (1998)
5. Sayegh, S. I.: Fingering for String Instruments with the Optimum Path Paradigm. Comput. Music J. 13, 76 – 84 (1989)
6. Parncutt, R., Sloboda, J.A., Clarke, E.F., Raekallio, M., Desain, P.: An Ergonomic Model of Keyboard Fingering for Melodic Fragments. Music Percept. 14, 341–382 (1997)
7. Parncutt, R.: Modeling Piano Performance: Physics and Cognition of a virtual Pianist. In: Proceedings of Int. Computer Music Conference, pp. 15–18 (1997)
8. Viana, A.B., de Morais Júnior, A.C. Technological Improvements in the SIEDP. In: IX Brazilian Symposium on Computer Music, Campinas, Brazil (2003)
9. Hart, M., Bosch, R., Tsai, E.: Finding Optimal Piano Fingerings. The UMAP Journal, 2, 167–177 (2000)
10. Radicioni, D.P., Anselma, L., Lombardo, V.: An Algorithm to Compute Fingering for String Instruments. In: Proceedings of the National Congress of the Associazione Italiana di Scienze Cognitive, Ivrea, Italy (2004)

11. Al Kasimi, A., Nichols, E., Raphael, C.: A Simple Algorithm for Automatic Generation of Polyphonic Piano Fingerings. In: 8th International Conference on Music Information Retrieval, Vienna (2007)
12. Yonebayashi, Y., Kameoka, H., Sagayama, S. Automatic Decision of Piano Fingering Based on Hidden Markov Models. In: IJCAI, pp. 2915–2921 (2007)
13. Jacobs, J.P.: Refinements to the Ergonomic Model for Keyboard Fingering of Parncutt, Sloboda, Clarke, Raekalliio and Desain. Music Percept., 18, 505–511 (2001)
14. Lin, C.-C., Liu, D.S.-M.: An Intelligent Virtual Piano Tutor. In: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications, pp. 353-356 (2006)
15. Balliauw, M.: A Variable Neighbourhood Search Algorithm to Generate Piano Fingerings for Polyphonic Sheet Music. Master's thesis at University of Antwerp, Faculty of Applied Economics, Antwerp (2014)
16. Sébastien, V. Ralambondrainy, H., Sébastien, O. Conruyt, N.: Score Analyzer: Automatically Determining Scores Difficulty Level for Instrumental E-learning. In: ISMIR, pp. 571-576 (2012)
17. Sörensen, K., Glover, F.: Metaheuristics. In: Glass, S.I., Fu, M.C. (eds.) Encyclopedia of Operations Research and Management Science, pp. 960–970 (2013)
18. Herremans, D., Sörensen, K.: Composing First Species Counterpoint with a Variable Neighbourhood Search Algorithm. Journal of Mathematics and the Arts, 6, 169–189 (2012)
19. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (1993)
20. Mladenović, N., Hansen, P.: Variable Neighbourhood Search. Computers and Operations Research, 24, 1097–1100 (1997)
21. Fiechter, C.-N.: A parallel tabu search algorithm for large travelling salesman problems. Discrete Appl. Math. 51, 243 – 267 (1994)
22. Gendreau, M., Hertz, A., Laporte, G.: A Tabu Search Heuristic for the Vehicle Routing Problem. Manag. Sci. 40, 1276 – 1290 (1994)