A variable neighbourhood search algorithm to generate piano fingerings for polyphonic sheet music

Matteo Ballliauw[1], Dorien Herremans[1], Daniel Palhazi Cuervo[1], and Kenneth Sörensen[1]

[1]University of Antwerp, Prinsstraat 13, 2000 Antwerp, Belgium

Abstract

A piano fingering indicates which finger should play each note in a piece. Such a guideline is very helpful for both amateur and experienced players in order to play a piece fluently. In this paper, we propose a variable neighbourhood search algorithm to generate piano fingerings for complex polyphonic music, a frequently encountered case that was ignored in previous research. The algorithm takes into account the biomechanical properties of the pianist's hand in order to generate a fingering that is user-specific and as easy to play as possible. An extensive statistical analysis was carried out in order to tune the parameters of the algorithm and evaluate its performance. The results of computational experiments show that the algorithm generates good fingerings that are very similar to those published in sheet music books.

Keywords: Metaheuristics, OR in music, Piano fingering generation, Variable neighbourhood search, Combinatorial optimisation

Author's version. The official version appears in print in International Transactions of Operations Research. October 2015.

1 Introduction

The piano is an instrument with one of the widest ranges of playable notes, spanning over seven octaves. It offers tremendous musical opportunities to both the pianist and composer. Unfortunately, a pianist only has ten fingers to play the piano's 88 keys. Therefore, it can be very helpful to have a piano fingering available, especially for beginner pianists. A *piano fingering* indicates which finger should play each note in a piece. Any finger could be used to play any note on a piano, contrary to instruments such as the saxophone or the recorder for which the fingerings are fixed. Depending on the melodic and harmonic properties of piano pieces, some fingerings are easier or better suited than others. The quality of a piano fingering depends to some degree on the pianist's expertise level and biomechanical properties (e.g., the maximum distance between two notes that can be played with the same hand varies widely between pianists), as well as the composer's desired interpretation (e.g., accented notes should be played with 'strong' fingers) (Sloboda et al., 1998).

There is a widespread agreement that a good piano fingering is crucial to allow a pianist to play a piece fluently (Parncutt et al., 1997). However, deciding on the right fingering for a piece of music can be a complex, time consuming and burdensome task (Hart et al., 2000). Automation of this process could help pianists, both novices and experts (Yonebayashi et al., 2007a). For the former, finding a fingering that allows them to play a piece fluently is difficult in its own right. The latter could save valuable time by having an automatically determined fingering, possibly consisting of different alternatives with different expressive characteristics (Gellrich & Parncutt, 1998; Parncutt et al., 1999).

Generating a piano fingering for a specific piece can be seen as a combinatorial optimisation problem. This is characterised by discrete decision variables that assign a specific finger to each note, an objective function that indicates the difficulty of the fingering, and a set of constraints to ensure the fingering is playable by a human. In this paper, we formulate the generation of a piano fingering as such a combinatorial optimisation problem and propose an effective and efficient algorithm to solve it. This algorithm and its parameter settings are thoroughly tested in two computational experiments. The approach proposed in this paper goes further than the current state-of-the-art in that it is able to handle complex polyphonic music, determines a fingering for both hands and allows for user-specific inputs.

In the next section, an overview of the current literature available for the generation of a piano fingering is discussed. In Section 3, finding a piano fingering for polyphonic music is defined as a combinatorial optimisation problem and a metric for evaluating the difficulty of a fingering is described. In Section 4, the developed variable neighbourhood search algorithm is discussed in detail. Section 5 evaluates the performance of the algorithm and describes the computational experiments carried out to tune its parameters. Section 6 shows example outputs and discusses the results. Conclusions and recommendations for further research are given in Section 7.

2 Literature review

The first exercises for learning how to select piano fingerings were published in the 18th century. C.P.E. Bach was one of the first to publish such work. He was followed by many others in the 19th century (Gellrich & Parncutt, 1998). The first mathematical translations and algorithm developments for this problem were proposed at the end of the 20th century. Similar research for other instruments can be found: e.g., automatic string instrument fingering by Radicioni et al. (2004).

2.1 Measuring the quality of a piano fingering

Throughout history, different *rules* emerged for deciding on an appropriate piano fingering. They even became repertory dependent (Clarke et al., 1997; Gellrich & Parncutt, 1998). According to Parncutt et al. (1997), the quality of a fingering can be measured by three objectives. The first objective is the *ease of playing*, which is related to the size and biomechanics of the hand. A second objective is the *ease of memorisation*, as some musicians can memorise certain combinations more easily than others. Finally, the third objective is *musical expressiveness*. A good fingering should help the pianist to convey the musical message intended by the composer.

In order to comply with the first objective, any algorithm that generates piano fingerings should take personal biomechanics of a pianist into account. Robine (2009) shows that the chosen fingering influences the musical execution and expressiveness, as considered by the third objective. This author gives the specific example of small lags between two notes, caused by passing the thumb underneath another finger. Because of the aforementioned, different pianists choose different fingerings for the same piece, as proven by Parncutt et al. (1997, p. 369-372). This latter criterion is especially important for more advanced pianists, since the potential to express certain musical interpretations with different fingering combinations becomes possible. When deciding on a fingering, a trade-off between these three criteria should be made (Clarke et al. 1997; Sloboda et al. 1998, p. 185). According to Clarke et al. (1997, p. 100) however, some details of fingerings are subject to personal preferences. As a result, we believe that they cannot all be captured by the model and that manual adaptations afterwards may be required.

Quantifying the *quality of a fingering* is one of the most important aspects in defining the generation of a piano fingering as a combinatorial optimisation problem. One of the first papers to introduce a cost-minimising paradigm to quantify the difficulty of a piano fingering in the literature is due to Parncutt et al. (1997). The higher the cost of a fingering, the more difficult it is to play the piece using the specified fingers. The authors increase the cost of a fingering based on rules that penalise difficult sequences and distances in a fingering. This set of rules is also applied

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

by Parncutt (1997) and Lin & Liu (2006) and expanded by Jacobs (2001). An important feature of this *cost minimising paradigm*, is that it uses a set of complementary rules. This makes it possible to identify certain trade-offs between different origins of difficulty and assign a weight to them. The rules from this approach are interpretable and have a musical/biomechanical origin, contrary to machine learning methods that learn a transition matrix based on a monophonic fragment (Robine, 2009; Hart et al., 2000) or short fragments with simple polyphonic chords (Al Kasimi et al., 2005, 2007). Yonebayashi et al. (2007a) learned Hidden Markov Models [HMM], whereby the fingering with the highest probability is considered to be the best alternative. Nakamura et al. (2014) implement an HMM for simple polyphonic music. They state that the advantage of this approach is that it differentiates between the left and right hand without needing a predetermined separation between notes in the right and left hand staff.

2.2 Solution strategies

The solution strategies found in the literature for finding good piano fingerings differ widely. Parncutt et al. (1997), Al Kasimi et al. (2005, 2007) and Lin & Liu (2006) make use of the Dijkstra algorithm. This algorithm calculates the shortest path in a graph, by retaining the minimal total cost paths through nodes containing the previous, actual and following used finger. This total cost is the accumulation of all the costs of the previous nodes on this path, increased with the cost of the actual node. At the last node of the algorithm, the shortest path is found. Moreover, as this algorithm was implemented on a situation which excluded a considerable amount of solutions which could not be played legato[1] from the solution space, Parncutt et al. (1997) claimed that its execution time had been reduced from exponential to polynomial. This approach however is not well suited to deal with complex polyphonic music whereby simultaneous notes have different start and end points, because this would result in an exponentially large number of nodes. It could also be possible that a complex polyphonic piece cannot or should not be played legato. A similar algorithm would then result in an empty solution space. In general, complex polyphony with different simultaneous melody lines complicates the application of any algorithm using networks or graphs.

Hart et al. (2000) and Robine (2009) use dynamic programming algorithms with very similar characteristics to the Dijkstra algorithm. This alternative approach finds the shortest path in a graphical network, using transition matrices and calculating backwards from the end to the beginning. In each node n, the finger used for the n-th note is indicated. For the last node, the cheapest transition to reach this node from all previous possible nodes is calculated. This is repeated for the previous node, considering the sum of the previously calculated score of the best transitions to the last node and the score of this additional transition. The algorithm calculates its way back to the first node, by repeating this action $N-1$ times with a total of N nodes. The first node corresponding to the lowest total cost, is chosen as the solution. From then on, the best transitions have to be chosen from the previously explored transitions. Similar to this approach, Yonebayashi et al. (2007a) and Nakamura et al. (2014) use the Viterbi Algorithm [VA] to find the optimal solution from a HMM for monophonic music. As Forney Jr (1973) states, VA is a simplified version of forward dynamic programming to find the shortest path in a graph that corresponds to the maximal a posteriori probability [MAP] estimation problem for a sequence of states, such as a HMM.

3 Problem description

In this section, the problem of finding an optimal fingering is presented as a combinatorial optimisation problem. The decision variables are discussed below, followed by a description of the objective function and imposed constraints. At the end of this section, the complexity of the problem is discussed.

[1] smoothly connected

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

3.1 Decision variables

A piano fingering indicates which finger should be used to play each note. In piano music, a piece consists of two staves, whereby the notes from the upper staff are played by the right hand and those from the lower staff by the left hand. This means that a solution of the piano fingering problem can be split up in two different sub problems, one for every hand or staff.

The convention when writing fingerings is to label every finger with a number from one up to five, where the first finger is the thumb. This is illustrated in Figure 1.

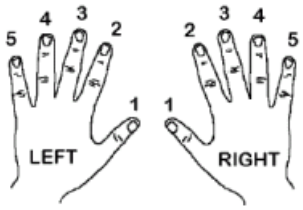


Figure 1: Convention for piano fingering representations (Yonebayashi et al., 2007b).

3.2 Objective function

In this paper, we define a set of rules that is used to evaluate the difficulty of a candidate fingering. This is done by measuring the extent to which the rules are followed. In analogy with Parncutt et al. (1997), Jacobs (2001) and Al Kasimi et al. (2005), an objective function based on an adapted version of their rules is minimised. This function penalises difficult combinations of fingers. In order to properly evaluate biomechanical characteristics of finger positions, a distance matrix is first defined.

3.2.1 Distance matrix

How easy is it to play two subsequent notes with a particular set of fingers? This depends mainly on the finger pair used and the physical distance between the notes on the piano. When using two adjacent fingers, the distance between the notes should be small. To quantify this idea, a distance matrix is defined. The convention of Parncutt et al. (1997) is used, in which six different types of distances are defined per finger pair. `MinRel` and `MaxRel` stand for minimal and maximal relaxed distances, which means that two notes separated by this physical distance on the keyboard can easily be played with this finger pair, while still maintaining a relaxed hand. `MinPrac` and `MaxPrac` define the largest distances that can be played by each finger pair. All of these distances are user-specific and can easily be adapted in the implementation of the described algorithm. The two final distances are `MinComf` and `MaxComf`. These show the distances that can be played comfortably, in the sense of not having to stretch to a maximal spread. They depend on the values of `MinPrac` and `MaxPrac`, as can be derived from Eq. (1) (Parncutt et al., 1997).

$$\begin{aligned} \texttt{MaxComf}o &= \texttt{MaxPrac} - 2 \text{(for all finger pairs).} \\ \texttt{MinComf} &= \texttt{MinPrac} + 2 \quad \text{(for finger pairs including the thumb),} \\ &= \texttt{MinPrac} \quad \text{(for finger pairs without a thumb).} \end{aligned} \tag{1}$$

When evaluating a fingering, ranges of distances are required in order to know which are the largest spreads that can be attained. For this reason minimal and maximal distances are defined to delimit the relaxed, comfortable and practical ranges.

In this paper, distances are expressed in *units* that measure how many piano keys one note is separated from another. This is analogous to the definition of Parncutt et al. (1997) and is visualised in Figure 2. Nevertheless, distances between some keys are altered in this research in order to solve the issue noticed by Jacobs (2001). He stated that a missing black key between

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

Table 1: Example distance matrix for a large right hand (adapted from Parncutt et al. (1997)).

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

E and F on one hand, and between B and C on the other hand, does not influence the distance between these respective key pairs. Therefore we introduce two imaginary, unused keys (key 6 and 14 in Figure 2) where a black key is missing. The numbering of the keys (or relevant pitches) continues in the next octave, in such a way that the next key (C) would be attributed a value of 15. This representation system has several advantages. First, the distance between two keys can now correctly be calculated without a bias for the 'missing keys'. Moreover, by calculating modulo 2, one can distinguish between black and white keys. Finally, by calculating modulo 14, the pitch class can be decoded.

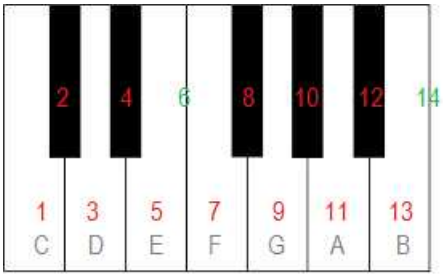


Figure 2: Distances on a piano keyboard with additional imaginary black keys.

All finger pairs involving the same finger twice (e.g., fingers 1-1) are set to zero in the distance matrix. An example of the other values in such a distance matrix for a large right hand is given in Table 1 as an adaptation of Parncutt et al. (1997), taking into account the newly defined keyboard distances. For pianists with smaller hands, tables with reduced absolute values of `MinPrac`, `MaxPrac`, `MinRel` and `MaxRel` are available in the developed software. Alternatively, users could adapt them according to their personal preferences, as was also suggested by Al Kasimi et al. (2005, 2007).

Interpreting the values in Table 1 is straightforward in combination with the keyboard image in Figure 2. A value of -10 for `MinPrac_R`(1-2) means that when the thumb is put on A4 (11), the index of a large right hand can be stretched no further than C4 (1), because $1 - 11$ equals -10. In order to calculate the distances for the finger pair where the order of the fingers used in first and second position are swapped, `Min` and `Max` have to be interchanged and the values have to be multiplied by -1. E.g., to calculate `MinPrac_R`(2-1) from finger 2 to 1 of the right hand: take `MaxPrac_R`(1-2) (11) and multiply by -1. This gives `MinPrac_R`(2-1) $= -11$. To deduce the information for the left hand, the order of the finger pair needs to be swapped (e.g., 1-2 becomes 2-1). E.g., to calculate `MinPrac_L`(2-1) = `MinPrac_R`(1-2) $= -10$ (Parncutt et al., 1997).

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

3.2.2 Rules

The previously defined distance matrix is used to calculate how well a fingering adheres to the set of fingering rules. These rules are listed in Table 2, which includes a description, the associated penalty in the objective function and the originating source for each rule. The *application* column specifies the relevance of each rule in the context of monophonic, polyphonic or both types of music.

Rules 1, 2, and 13 take the distance matrix into account for consecutive notes. These rules can be applied for each pair of subsequent notes both for monophonic and polyphonic music. To prevent unnatural finger crossings in chords, rules 1, 2, and 13 are also applied *within* a chord in rule 14 to avoid awkward positions. An example of such a position would be the right index finger placed on a lower note than the right thumb within one chord. Hence within rule 14, rules 1 and 2 are applied with doubled scores to account for the importance of this natural hand position within one chord.

It is important to point out that Parncutt et al. (1997) use the distances `MinPrac` and `MaxPrac` in rule 13 as hard constraints. These minimal and maximal distances of each finger pair are not to be violated. If no feasible fingering is found, then that particular (monophonic) part cannot be played legato. In this paper, these practical distances are implemented as soft constraints in the objective function, but they are assigned very high penalties per unit of violation (e.g., +10). This allows to apply the set of rules of Parncutt et al. (1997) on polyphonic music more easily. Otherwise, it would be very hard for the algorithm to find a feasible initial solution.

Rules 3 and 4 also consider the distance matrix and account for hand position changes, which have to be reduced to necessary cases only (Parncutt et al., 1997). Rules 5 up to 11 prevent difficult transitions in monophonic music. The original scores of rule 8 and 9 could respectively add up to five or four between one pair of notes, which is relatively high compared to a score of one or two, resulting from the application of the other rules. Therefore, the score obtained by these rules is divided by two compared to the original suggestion made by Parncutt et al. (1997). In addition, the use of the fourth finger should not always be avoided, since the pianist might want to train this finger. Hence the user is advised to change the score in rule 5 to zero, which is also the default setting for our experiments. However, since the use of finger three and four in any consecutive order is complicated by their biomechanical connection, rules 6 and 7 are retained in order to prevent this difficult combination (Leijnse, 2014).

Rule 12 prevents the repetitive use of a finger in combination with a hand position change. For instance, a sequence C4–G4–C5 fingered 2–1–2 in the right hand forces the pianist to reuse the second finger very quickly. This becomes problematic in fast pieces. Therefore, the finger combination 2-1-3, which is favoured by rule 12, offers a better solution. Rule 15 tries to prevent hand position changes when notes with the same pitch are played consecutively. When this rule would not be implemented, consecutive notes which have the same pitch could be played by a different finger.

The logic of the rules and their relative scores were obtained from discussions with professional musicians (Eeman, 2014; Swerts, 2014; Chew, 2014; Leijnse, 2014). The scores resulting from the application of each rule are summed for each hand to form the total difficulty score per hand for a candidate fingering. This is shown in Eq. (2). This sum forms the objective function of the piano fingering problem and should be minimised in order to find an easily playable fingering. For each hand, one objective function is defined, because each hand can be optimised independently.

$$\min f(s) = \sum_i f_i(s) \tag{2}$$

An advantage of this approach is that it allows the user to change the score of different rules and thus change their relative importance. For instance, pianists who do not like to use the fourth finger might set the score of rule 5 back to one instead of zero.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

Table 2: Set of rules composing the objective function (adapted from Parncutt et al. (1997), Jacobs (2001) and Al Kasimi et al. (2007)).

Rule	Application	Description	Score	Source
1	All	For every unit the distance between two consecutive notes is below `MinComf` or exceeds `MaxComf`.	+2	Parncutt et al.
2	All	For every unit the distance between two consecutive notes is below `MinRel` or exceeds `MaxRel`.	+1	Parncutt et al. and Jacobs
3	Monophonic	For three consecutive notes: If the distance between a first and third note is below `MinComf` or exceeds `MaxComf`: add one point. In addition to that, if the pitch of the second note is between the other two pitches, 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	Parncutt et al.
4	Monophonic	For every unit the distance between a first and third note is below `MinComf` or exceeds `MaxComf`.	+1	Parncutt et al.
5	Monophonic	For every use of the fourth finger.	+1	Parncutt et al. and Jacobs
6	Monophonic	For the use of the third and the fourth finger consecutively.	+1	Parncutt et al.
7	Monophonic	For the use of the third finger on a white key and the fourth finger on a black key consecutively in any order.	+1	Parncutt et al.
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	Parncutt et al.
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	Parncutt et al.
10	Monophonic	For a thumb crossing or passing another finger on the same level (white–white or black–black).	+1	Parncutt et al.
11	Monophonic	For a thumb on a black key crossed by a different finger on a white key.	+2	Parncutt et al.
12	Monophonic	For a different first and third consecutive note, played by the same finger, and the second pitch being the middle one.	+1	Own rule
13	All	For every unit where the distance between two following notes is below `MinPrac` or exceeds `MaxPrac`.	+10	Own rule, based on constraint of Parncutt et al.
14	Polyphonic	Apply rules 1, 2 (both with doubled scores) and 13 within one chord.		Own rule, based on Al Kasimi et al.
15	All	For consecutive slices containing exactly the same notes (with identical pitches), played by a different finger, for each different finger.	+1	Own rule

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

3.3 Constraints

In addition to the soft constraints defined in the objective function above, a hard constraint is defined, stating that the same finger cannot be used on two different keys played at the same time. A violation of this constraint results in a rejection of the solution.

3.4 Complexity of the problem

The piano fingering problem can be considered as a special case of the partial constraint satisfaction problem [PCSP] (Koster et al., 1998). The PCSP is defined by a so-called constraint graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Each vertex $v \in V$ represents a decision variable that must be assigned a value from a given domain D_v. Each value in D_v has a penalty associated to it, which is defined by a vertex-penalty function $Q_v : D_v \to \mathbb{R}$. An edge $\{v, w\} \in E$ indicates that there are penalties attached to certain combinations of values for vertices v and w. These penalties are defined by an edge-penalty function $P_{\{v,w\}} : \{\{d_v, d_w\} \mid d_v \in D_v, d_w \in D_w\} \to \mathbb{R}$. In a more formal notation, the PCSP can be defined by the quadruple $(G = (V, E), D_V, P_E, Q_V)$, where D_V is the set of domains D_v (for each vertex $v \in V$), P_E is the set of edge-penalty functions $P_{\{v,w\}}$ (for each edge $\{v, w\} \in E$) and Q_V is the set of vertex-penalty functions Q_v. The objective of the PCSP is to find an assignment $A = \{a_v \mid v \in V, a_v \in D_v\}$ that minimizes the total sum of the penalties $\sum_{\{v,w\}\in E} P_{\{v,w\}}(\{a_v, a_w\}) + \sum_{v\in V} Q_v(a_v)$. This problem is known to be NP-hard as it can be reduced to the maximum satisfiability problem (Koster et al., 1998, 1999).

The piano fingering problem can be modelled as an instance of the PCSP in which each note in a piece is represented by a vertex $v \in V$. In this sense, each vertex (note) must be assigned a value (finger) from a common domain $D_v = \{1, 2, 3, 4, 5\}$. The set of rules defines the edges that connect each pair of vertices $\{v, w\} \in E$, the edge-penalty functions in P_E and the vertex-penalty functions in Q_V. Two vertices are connected by an edge if the notes they represent are played simultaneously or consecutively[2]. This condition causes the graph G that defines the PCSP instance to have a very specific structure. Nonetheless, to the best of our knowledge, it is unclear whether such properties make this kind of instances easier to solve.

4 Variable neighbourhood search algorithm

Exact algorithms are often used to solve a wide range of combinatorial optimisation problems. These algorithms guarantee to find the optimal solution. However, for complex problems, their execution time can grow exponentially with the size of the instance. For these cases, metaheuristics present a good alternative solution technique. Although they are not guaranteed to find the optimal solution, they tend to find very good solutions for complex instances in a reasonable execution time.

A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms (Sörensen & Glover, 2012). Heuristics are based on rules of thumb and are a frequently applied alternative to solve complex combinatorial optimization problems. Metaheuristics are divided into three categories; constructive, population-based and local search [LS] (Sörensen & Glover, 2012; Herremans & Sörensen, 2013). LS metaheuristics form a large class of metaheuristics that are characterised by the fact that they iteratively improve a solution (called the *current solution*) by applying small adaptations (*moves*) to it. For most representations of combinatorial optimisation problems, different *move types* can be defined. The set of all solutions reachable from a given current solution by a single move of a given move type is called this solution's *neighbourhood* (Sörensen & Glover, 2012). When no more improving moves of a certain type exist in the neighbourhood of the

[2]Note that this model of the piano fingering problem excludes complex rules that involve triplets of notes (like, for example, Rule 4 in Table 2). Nevertheless, such simplification is still useful to illustrate the complexity of the problem.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

current solution, a *local optimum* is reached. A possible way to escape such a local optimum is by using a different move type, a strategy commonly called *variable neighbourhood search* [VNS]. This strategy is based on the fact that a local optimum relative to a certain move type is not necessarily a local optimum relative to another (Mladenović & Hansen, 1997). When all neighbourhoods are exhausted, a *perturbation* randomly changes part of the best solution so that the next iteration of the algorithm can continue the search (Lourenço et al., 2003).

The algorithm developed in this paper is a VNS algorithm. Herremans & Sörensen (2012) also successfully applied a VNS algorithm to a musical problem, namely to the automatic composition of counterpoint music. A local search metaheuristic might also prove to be successful in piano fingering, as the process of making small changes to a fingering is analogous to an artist making final adaptations to a given fingering (whether or not indicated by the composer). This approach is similar to how Sloboda et al. (1998, p. 185) describe the human process involving some heuristics. Moreover, since metaheuristics can generate good-quality solutions in a reasonable execution time, they offer a promising optimisation framework to generate good fingerings for complex polyphonic pieces.

The implemented VNS algorithm does not require a graph based representation for solutions, which allows it to be easily applied to polyphonic music. The complex nature of polyphonic music makes it very difficult and computationally expensive to use an exact approach. Metaheuristics offer an efficient way of solving the specified piano fingering problem.

As can be seen from the flowchart diagram in Figure 3, the algorithm starts by generating a random initial solution. Every note is assigned a finger at random. In order to obtain a feasible solution, once a finger is used in a particular time *slice*, this finger can no longer be assigned to another note in the same slice. A slice is a part of the piece with the duration of the shortest note from the piece. In this way, the duration of a note consists of an integer number of slices. The described process works its way through the piece from left to right. Assuming that maximum five notes are to be played at the same time by one hand, a feasible solution is guaranteed. After generating the initial fingering for both staves, the right and left hand are optimised separately. An initial preprocessing step, called Swap, is performed to make a large improvement by interchanging all assignments of two fingers throughout the entire piece. This step produces a relatively small neighbourhood with a maximum size of 10 possible moves[3].

The implemented local search procedure (VNS) uses different neighbourhood structures, each defined by a move type. In order to get an indication of the complexity of the algorithm, the formulas for calculating the size of each type of neighbourhood are given below. These formulas are based on a piece containing chords, each with the same amount of notes. t is used to indicate the number of notes in the piece and s refers to how many notes are played simultaneously within one chord. The neighbourhood sizes for the example piece in Figure 4 are calculated below.

In the algorithm, there are three neighbourhood structures included:

Change1 The Change1 move type changes the fingering of each note to any other allowed finger. The resulting neighbourhood contains feasible fragments in which one finger is changed. An example of this type of move is given in Figure 5(a). The size of this neighbourhood can be calculated as $(5-s)t$.
In the example piece in Figure 4, there would be 24 possible solutions in a Change1 neighbourhood.

Change2 For the Change2 move type, the previous Change1 move is expanded to changing the fingering of two notes that are either adjacent or simultaneously played. This results in a neighbourhood which consists of fragments with two changed fingers. For an example, we refer to Figure 5(b). The size of this neighbourhood is given by $\binom{s}{2}[(6-s)(5-s)+1]+(\frac{t}{s}-1)\{\binom{s}{2}[(6-s)(5-s)+1]+s^2(5-s)^2\}$.
In the example piece in Figure 4, this neighbourhood contains 192 solutions.

SwapPart A SwapPart move type is included to enhance the changeability of the fingering of simultaneous notes. Suppose a long note a, assigned finger f is played simultaneously with

[3] $\binom{5}{2}$

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

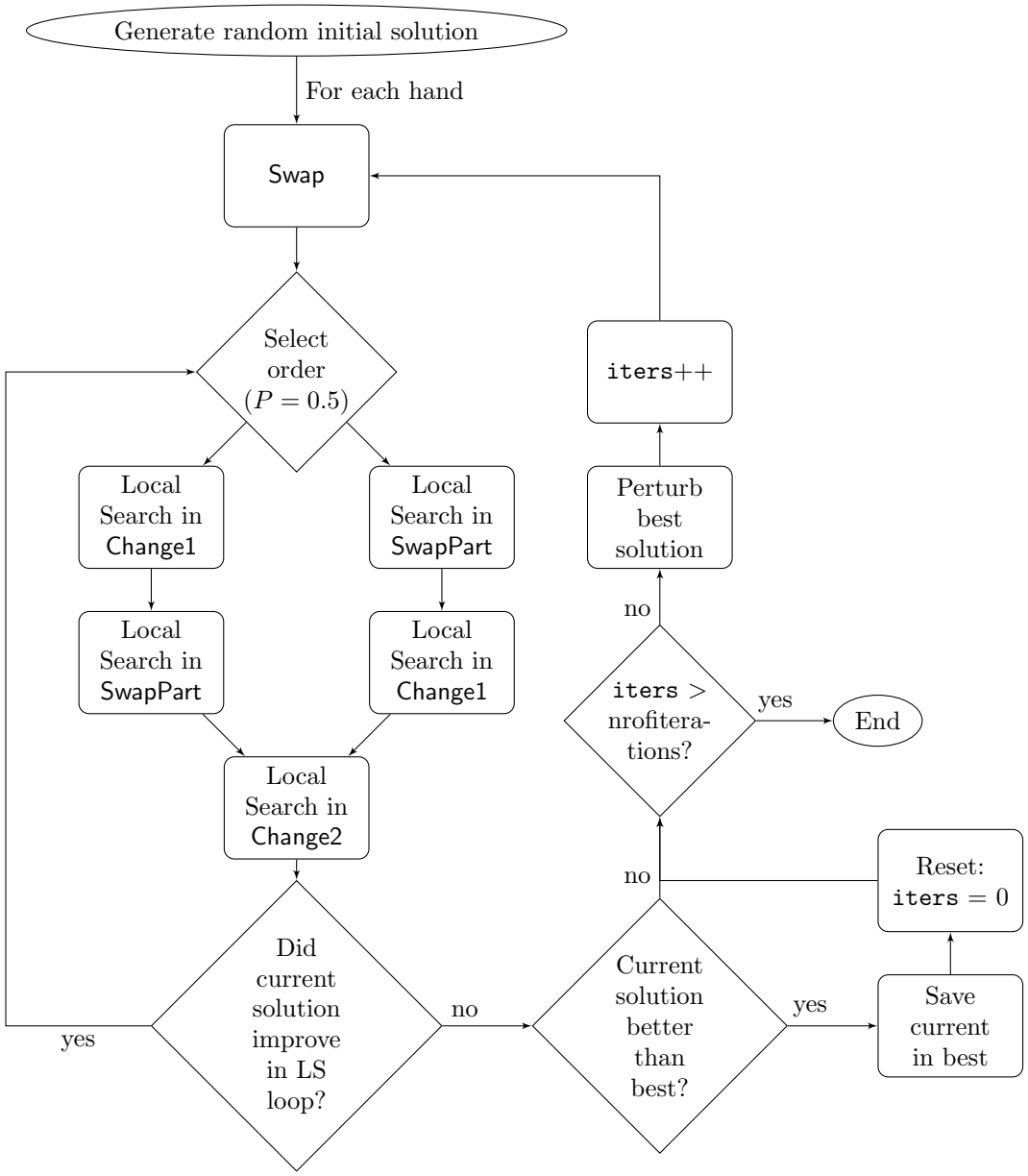


Figure 3: Structure of the algorithm.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

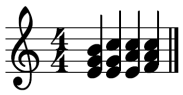
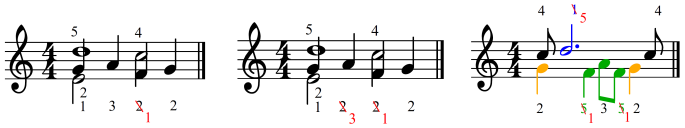


Figure 4: A short example piece used to calculate sizes of the neighbourhoods, with $s = 3$ and $t = 12$.

multiple short notes $b_1, b_2, ..., b_k$, each assigned the same finger g. Since all notes b_i are fully contained within note a, none of the previous move types would allow note a to be assigned finger g, as this would break the basic feasibility rule (because one finger would be playing multiple notes at the same time). The SwapPart however does allow the algorithm to swap fingers f and g, even in cases where it would otherwise not be possible without breaking feasibility. Moreover, this strategy only swaps the fingering of simultaneous notes that are fully contained within the interval of the longer note in order to reduce the probability of arriving at an infeasible solution. An example of this move type is shown in Figure 5(c). In the figure, D5 is note a, played by finger 1. Notes F4, A4 and again F4 are played simultaneously with D5. Both the F4's could be considered as notes b_1, b_2, played with finger 5. The fingering of the longer note can only be changed from 1 to 5 when all simultaneous notes played with finger 5 are changed to the previous fingering of the long note. This is exactly what the SwapPart move can do, it allows us to change the fingering of longer notes in a polyphonic piece. This neighbourhood has a size of $4t$.
A SwapPart neighbourhood from the example piece in Figure 4 would contain 48 moves.



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

Figure 5: Examples of a move from the different neighbourhoods.

In order to select a solution from a neighbourhood as the new current solution, a *move strategy* has to be selected. In this paper, two different strategies were implemented, namely steepest and first descent. In a steepest descent strategy, the entire neighbourhood is calculated before the solution with the biggest improvement is selected. In a first descent strategy, the first solution that improves the current solution is selected. In Section 5, we study the influence of the chosen move strategy on the solution quality and the execution time.

Figure 3 shows how the different neighbourhoods are chained in the VNS strategy. The algorithm continues to perform a LS in a neighbourhood until it reaches a local optimum. The algorithm then moves on to the next neighbourhood, and continues until no better solution is found. When the local optimum of the last neighbourhood in the chain (Change2) is reached, the algorithm starts from the first neighbourhood again. The flowchart shows a split after the initial Swap step. This indicates that the order of Change1 and SwapPart is chosen randomly (with a 50–50% probability) by the algorithm. The reason for this design choice is that the best order is fragment dependent. The performance of both neighbourhoods depends on the monophonic or polyphonic character of a fragment.

This search strategy continues until none of the three LS neighbourhoods can find a better solution. When such a local optimum is reached, this solution is compared to the best solution up to that point. When it is better, it is stored as the new best solution and the number of iterations without improvements is reset to zero. If the current solution is not better than the best solution, the number of iterations without improvement is compared with the stopping parameter (maximum allowed number of iterations without improvement). When the maximum number

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

of iterations without improvement is reached, the algorithm stops. Otherwise, a perturbation is performed on the best solution found so far. This new solution is then handed back to the Swap step and a new iteration of the algorithm is initialised.

The perturbation splits up the piece in different parts. A set of subsequent slices is selected in each part. The fingerings in the selected slices are replaced by new ones. The new random fingerings are chosen only from unused fingers per slice in order to guarantee the feasibility of the solution. The size of the parts and the set of slices within each part are defined as a percentage of the length of the piece and the number of slices in each part respectively. This approach is preferred over changing the fingering of random notes spread across the piece, because it is very likely that in polyphonic music such changes are infeasible. For example, if you have a chord of five notes, you have to remove at least the fingering of two notes in order to obtain a feasible perturbed solution. Hence, perturbing entire sets of consecutive slices results in the best option.

The VNS algorithm has been implemented in C++ and is available as Open Source software[4]. For inputting sheet music and outputting sheets with annotated fingerings the MusicXML format is used. This format was designed for easy interchangeability (Good, 2001). They can be read by most music notation software, including the open source program MuseScore[5], which was used to visualise the results in this paper.

5 Experiments

The algorithm developed in Section 4 brings up the need for decisions concerning the parameter setting of its different components (e.g., size of the perturbation) that ensure optimal performance. Two experiments were conducted in order to base the choice of these parameters on a thorough statistical analysis. First, we describe the experiment conducted to determine the settings that are used by the intensification phase of the VNS. Afterwards a second experiment was carried out to set the parameters of the entire VNS.

Independence of these two parameter groups is assumed, an approach similar to the one used by Palhazi Cuervo et al. (2014) in their experiments, focusing on a subset of parameters. The resulting best parameters of the first experiment are implemented in the second. By setting up the experiments like this, the computing time decreases. It is reasonable to assume that the performance of the intensification phase is independent of the entire VNS framework. Hence, the experiment can be split in two sub-experiments. As a result, instead of having to test $A \cdot B$ combinations, we only have to test $A + B$ combinations.

Both experiments were run ten times on ten pieces of one to three pages in sheet music notation. All experiments were run on a 2.93 GHz Intel Core i7 processor.

5.1 Intensification phase

The goal of the first experiment is to determine the optimal values for the parameters of the intensification phase. It takes five parameters into account. The four parameters `step_swap` and `nbh_X` indicate the inclusion (On) or exclusion (Off) of the improvement step or the respective neighbourhood in the algorithm. A final parameter `improv_strat` indicates the use of steepest descent or first descent as the improvement strategy. An overview of the tested parameter values is given in Table 3. The impact of these parameters on two dependent variables is studied: the solution quality and the computing time. The quality of the solution is measured through the summed scores of the objective functions for the left and right hand, denominated `Objective function`. The variable `Runtime` is equal to the total computing time (user time) used for one test run.

A full factorial experiment was conducted on ten different classical input pieces from varying style periods. For every piece, 2^5 or 32 executions of the algorithm were performed. This was

[4] http://dorienherremans.com/automatic-piano-fingerings
[5] Available from musescore.org.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

repeated ten times for every piece. Since we considered a benchmark set of ten pieces, a total of 3200 observations was obtained.

Table 3: Tested values of the parameters for the intensification phase experiment.

Parameter	Values
`step_swap`	Off, On
`nbh_change1`	Off, On
`nbh_change2`	Off, On
`nbh_swapp`	Off, On
`improv_strat`	Steepest, First

The data generated by the experiment was analysed by means of a mixed-effects analysis of variance (ANOVA) using the statistical software JMP. The model considers the name of the piece as a random effect and includes second degree interaction effects. The p-values of the F-tests listed in Table 4 indicate the significance of the impact of the different parameters. The table also includes parameters with interesting, significant interaction effects. This table shows that the inclusion of the preprocessing step and the activation of each neighbourhood has a significant impact on the performance of the algorithm in terms of the `Objective function`. The mean plots in Figure 6 show that including a component in the algorithm has a positive impact on its performance.

When examining the influence of the `improv_strat` parameter, it is important to notice that using steepest or first descent has no impact on the `Runtime`, nor on the `Objective function` (see Table 4). This was verified by a non-significant first order effect in the models where only first-order effects were included. When looking at the impact of some significant interaction effects on the execution time in the extended model (see Figure 7), it becomes clear that the impact of the improvement strategy on the `Runtime` is neighbourhood-dependent. In Change1, first descent appears to be the fastest strategy, but in Change2, steepest descent is significantly faster.

Figure 8 shows the evolution of the score over time of a run of the algorithm with steepest versus first descent. It is apparent that first descent requires more moves to arrive at a similar final solution, as the improvements are smaller than those of steepest descent. These smaller steps however generally take less time, as the algorithm does not have to search through the entire neighbourhood. As a result, first improvement improves faster at the beginning of a run. However, the described inverse relationship between the improving capacity of a move from an improvement strategy on one hand and and its time required on the other hand, causes both strategies to converge after a while and obtain similar results, which is confirmed by the high p-value of this parameter.

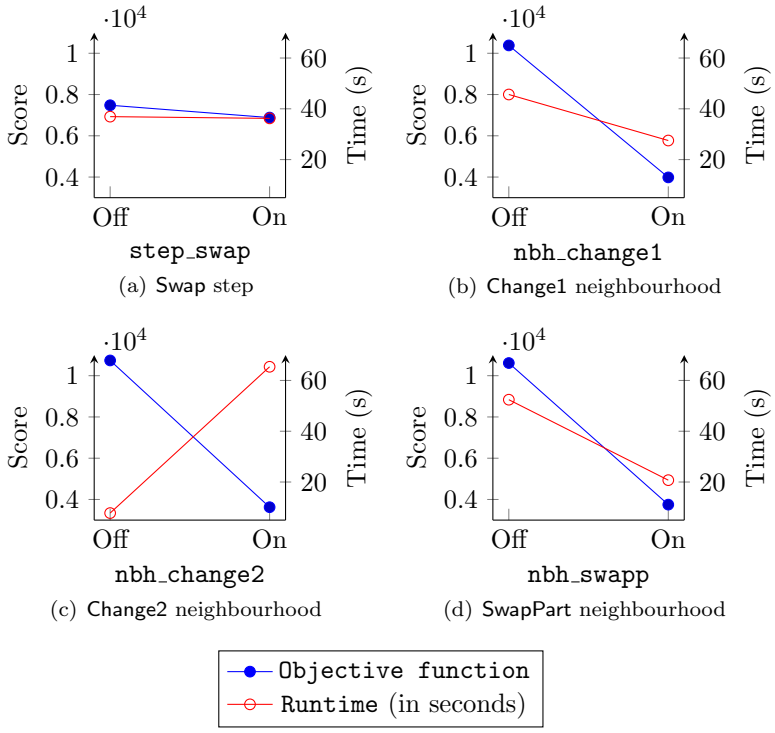
Based on the analysis above, the Swap step and every neighbourhood are included in the optimal design of the algorithm. The chosen improvement strategy is steepest descent, as it

Table 4: p-Values for an extract of the F-tests in the ANOVA model of `Objective function` and `Runtime` for the intensification phase experiment.

Parameter	`Objective function`	`Runtime`
`step_swap`	<0.0001 *	0.4927
`nbh_change1`	<0.0001 *	<0.0001 *
`nbh_change2`	<0.0001 *	<0.0001 *
`nbh_swapp`	<0.0001 *	<0.0001 *
`improv_strat`	0.6496	0.1273
`nbh_change1*improv_strat`	0.3960	<0.0001 *
`nbh_change2*improv_strat`	0.2772	0.0025 *

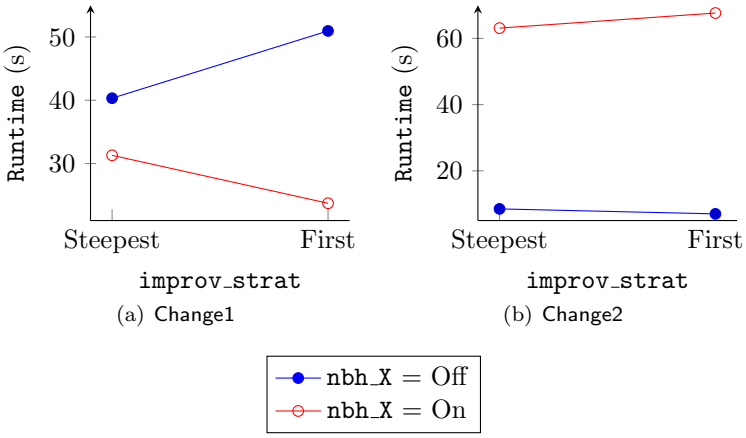
* significant at $\alpha = 0.05$

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.



(a) Swap step (b) Change1 neighbourhood (c) Change2 neighbourhood (d) SwapPart neighbourhood

Figure 6: Mean plots for the intensification phase experiment.



(a) Change1 (b) Change2

Figure 7: Interaction effects between different neighbourhoods and the improvement strategy for the intensification phase experiment.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

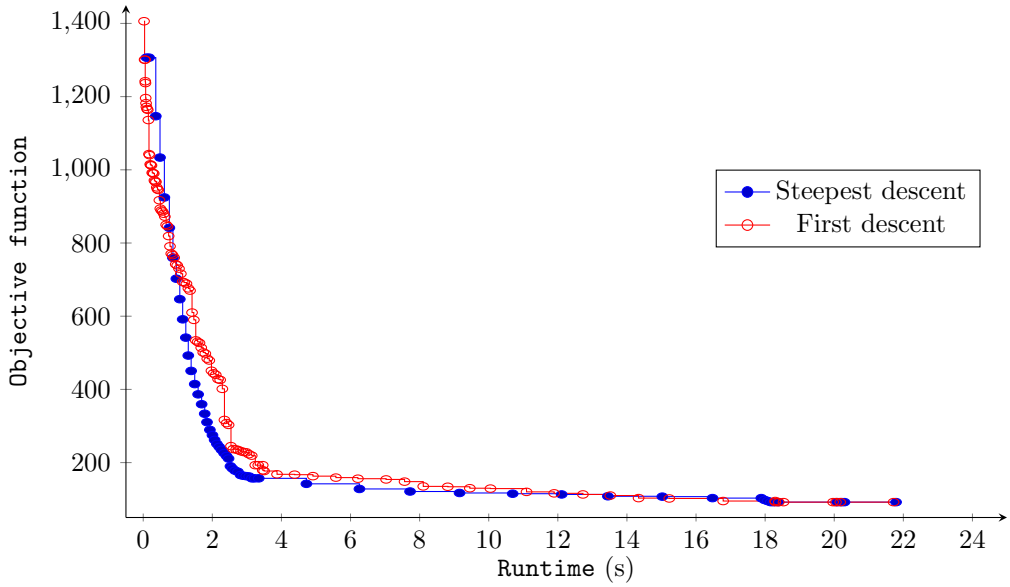


Figure 8: Evolution of the score over time with steepest and first descent.

appears to be slightly better and faster. An overview of these optimal values is given in Table 5.

Table 5: Best values of the parameters resulting from the intensification phase experiment.

Parameter	Value
`step_swap`	On
`nbh_change1`	On
`nbh_change2`	On
`nbh_swapp`	On
`improv_strat`	Steepest descent

5.2 Entire VNS

In this section, a second full factorial experiment is set up to identify the parameter settings related to the entire VNS. The same ten pieces are used and the optimal parameter settings of the previous experiment are used within the local search. Three parameters need to be set on the VNS level. The values tested in this experiment are displayed in Table 6. The number of allowed iterations without improvement is given by the parameter `nrofiterations`. The total amount of fingerings changed randomly by the perturbation is given by `sizeofpert` as a percentage of the total number of slices in a piece. The parameter `partsize` indicates the size of the perturbed parts as a percentage of the number of slices. A `partsize` of 25% e.g., results in four parts as described in Section 4. The studied dependent variables are the same as in the previous experiment, namely `Objective function` and `Runtime`. Preliminary tests showed that increasing `nrofiterations` beyond 10 and `sizeofpert` beyond 20 did not yield solutions of much better quality, therefore the parameters were cut off at these values. After running all 4^3 or 64 possible parameter combinations ten times on the ten different pieces, a total of 6400 observations was obtained.

Similar to the previous subsection, a mixed-effects ANOVA was performed in JMP. The instance name was defined as a random effect. Second degree interaction effects were included to see how the performance of the entire VNS is influenced by different combinations of parameters. Table 7 shows that the three tested parameters have a significant impact on both of the dependent

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

Table 6: Tested values of the parameters for the entire VNS experiment.

Parameter	Values
`nrofiterations`	1, 4, 7, 10
`sizeofpert`	5%, 10%, 15%, 20%
`partsize`	10%, 25%, 50%, 100%

Table 7: p-Values for the F-tests in the ANOVA model of `Objective function` and `Runtime` for the entire VNS experiment.

Parameter	`Objective function`	`Runtime`
`nrofiterations`	<0.0001 *	<0.0001 *
`sizeofpert`	<0.0001 *	<0.0001 *
`partsize`	<0.0001 *	<0.0001 *
`nrofiterations*sizeofpert`	<0.0001 *	<0.0001 *
`nrofiterations*partsize`	0.0918	0.0015 *
`sizeofpert*partsize`	0.6670	0.9252

* significant at $\alpha = 0.05$

variables (`Objective function` and `Runtime`). The impact of the parameter `partsize` is shown in a mean plot (Figure 9). The best performance is obtained with parts of 25%, or 4 parts per piece. Increasing the value of `nrofiterations` and `sizeofpert` will, as expected, result it solutions of better quality. The `Runtime` will however be higher. Table 7 confirms that the interaction effect between these two variables has a significant impact on `Objective function` and `Runtime`. The combined influence of both parameters on the score and time is visualised in Figure 10. One can see that the marginal performance enhancement of an increase in one of these two parameters is less when the other parameter already has a high value. Moreover, the required execution time would increase. As a result, increasing `nrofiterations` beyond ten and `sizeofpert` beyond 20% will not increase the quality of the solution by much, only the execution time would go up.

The best performing parameter settings resulting from the above analysis are ten iterations without improvement and four parts (each of 25% of the piece) of which 20% of the notes are perturbed with each iteration. These parameters are displayed in Table 8.

Table 8: Best values of the parameters for the entire VNS experiment.

Parameter	Values
`nrofiterations`	10
`sizeofpert`	20%
`partsize`	25%

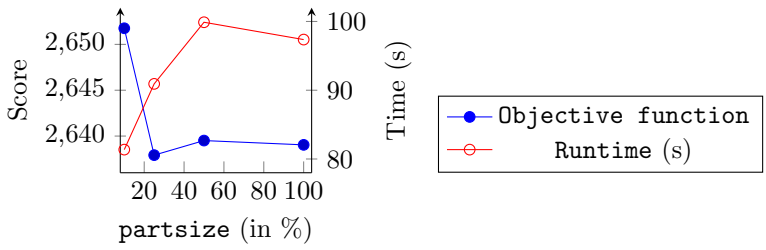
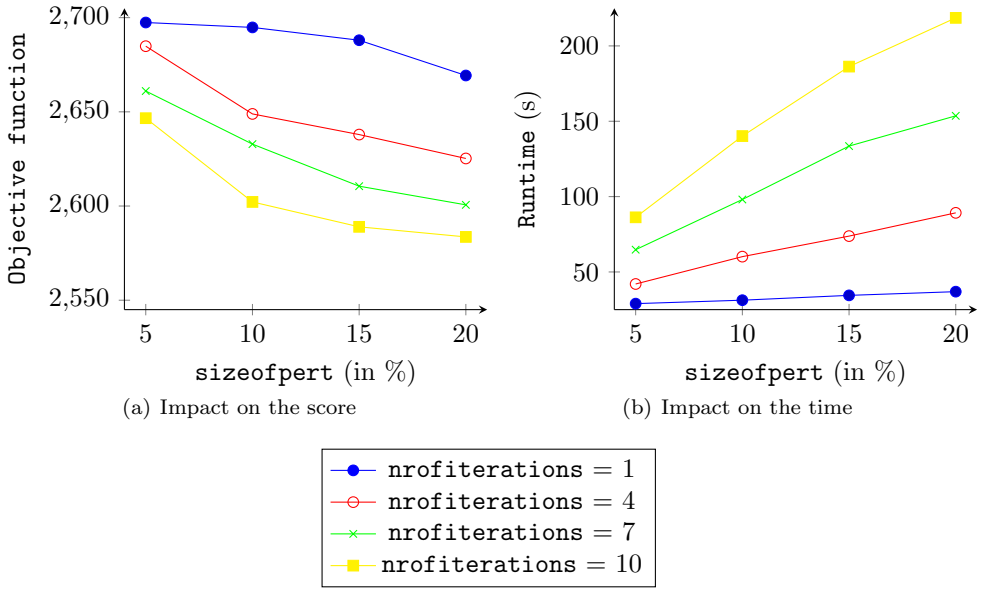


Figure 9: Mean plot of the size of the parts for the entire VNS experiment.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.



(a) Impact on the score

(b) Impact on the time

Figure 10: Interaction effect between the size of perturbation and the number of iterations for the entire VNS experiment.

6 Results

In this section, the performance of the VNS with the optimal parameter settings and its musical output are analysed. First, the output for a monophonic piece is discussed. This analysis is followed by the discussion of the output for polyphonic music. Finally, the difficulty of generated fingerings for respectively small, medium and large hands are compared.

6.1 Output for monophonic music

A fingering was automatically generated for Menuet 4 of Christian Petzold (formerly attributed to J.S. Bach) (BWV Anh. 114). This piece was not used during the tuning experiment in the previous section. The solution for both hands was computed in 110 seconds. The evolution of the score over time in the right hand is displayed in Figure 11. It shows that the perturbation strategy allows the algorithm to escape from local optima. After reaching a local optimum in one neighbourhood, new and better solutions become available in a previously exhausted neighbourhood, proving the effectiveness of the VNS strategy.

The output for the first eight bars of the piece is shown in Figure 12. The bold, circled fingerings are those that the composer or editor of a music sheet book prints on the sheets, so that the pianist automatically knows the rest of the fingering. In this paper, the fingerings are retrieved from sheets on IMSLP (2014). Compared to this given solution, in this excerpt only one finger was different from the one generated by the algorithm, in the sixth bar of the excerpt. In the remainder of this piece, a few other small sequences require manual changes.

6.2 Output for polyphonic music

A similar analysis was done based on a more complex, polyphonic piece, the first variation on the Saraband from G. F. Händels Suite in D minor (HWV 437). The final solution was computed in 38 seconds. This is less than the monophonic piece for two reasons. First, we see that this polyphonic piece is shorter than the monophonic piece, containing respectively 156 and 203 notes. Secondly, a comparison of the graphs in Figures 11 and 13 shows that the time-consuming operator Change2 requires more time in the monophonic piece.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

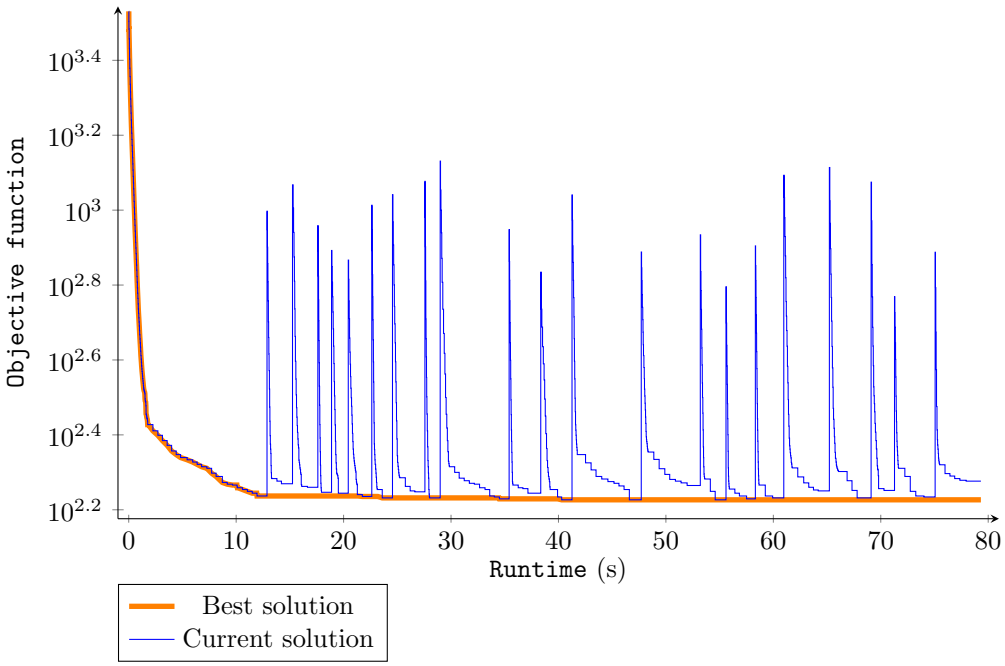


Figure 11: Evolution of the score over time in Menuet 4 (BWV Anh. 114) by Christian Petzold (formerly attributed to J.S. Bach).

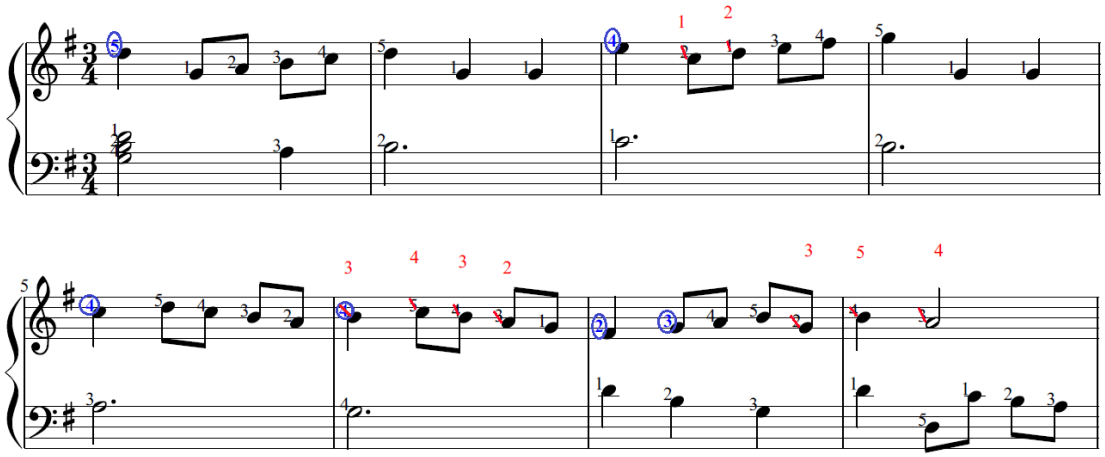


Figure 12: Output for the first eight bars of Menuet 4 (BWV Anh. 114) by Christian Petzold (formerly attributed to J.S. Bach).

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

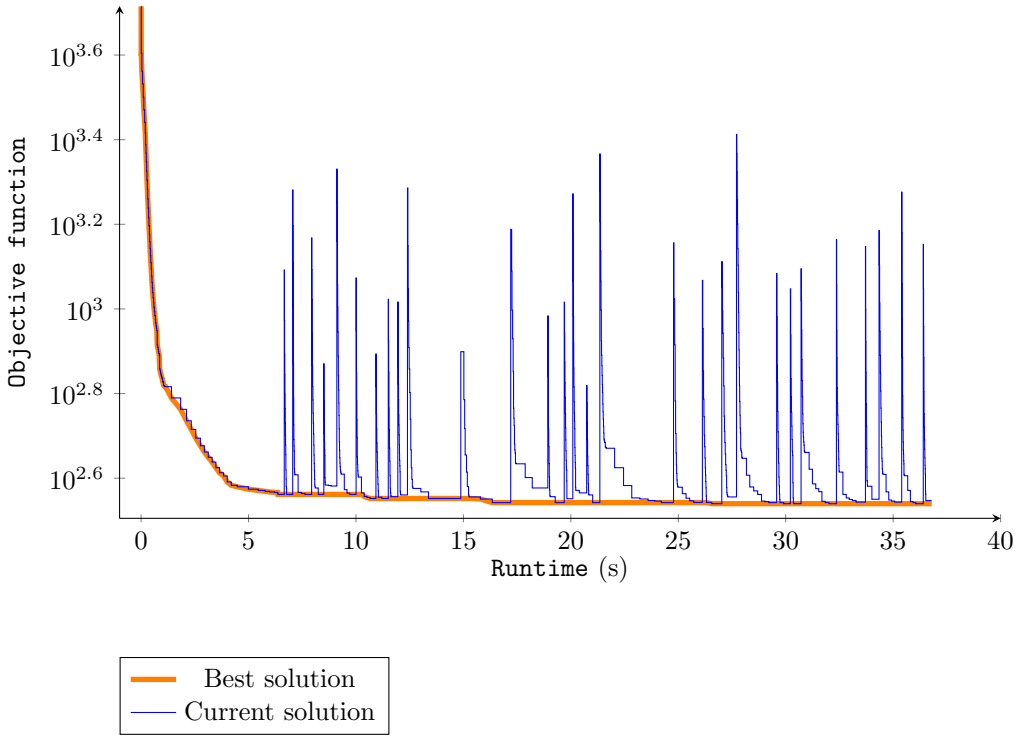


Figure 13: Evolution of the score over time in the first variation on the Saraband from Suite in D minor (HWV 437) by G.F. Händel.

The evolution of the score over time is included in Figure 13. The graph shows that better solutions keep being found, even after a long while. The output for the first eight bars of this piece is displayed in Figure 14. Only a few fingers (indicated in bold and circled) differ slightly from a printed version published on IMSLP (2014). The generated solution is fluently playable according the authors' subjective opinion, and only requires minor corrections. The output shows that only the fingers of short notes should be changed into adjacent fingers to obtain the fingering suggested by printed sources.

A well known piece of piano music which encompasses complex polyphony with multiple voices is the Contrapunctus XIV from J.S. Bach's Die Kunst der Fuge (BWV 1080), since it contains four simultaneous voices. Figure 15 displays the third movement, which is based on the BACH theme[6]. The conclusions from above remain valid in this complex piece. The sequence of the finger assignments follows the separate musical voices very well and there are only minor differences compared to the textbook fingerings.

An interesting issue comes up in bar 19, where the E4 in the second voice played on the second beat of the bar, cannot be played in one hand together with the A5. In practice, the E4 will be played by the thumb of the left hand. The assignment of different hands to a note is currently not included in the algorithm. This will be added in future versions of the software, as is discussed in the final section of this article.

6.3 Impact of hand size on fingering difficulty

The influence of the hand size on the difficulty of piano fingerings, measured by the objective function, is analysed in this subsection. Three types of hands were defined, a small, medium and large hand. The physical dimensions of a large hand are shown in the distance matrix in Table 1, while those for a small and a medium hand are included in Appendix A. The VNS was applied

[6]The first four notes are pronounced as BACH in German: B flat, A, C, B natural (H in German)

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

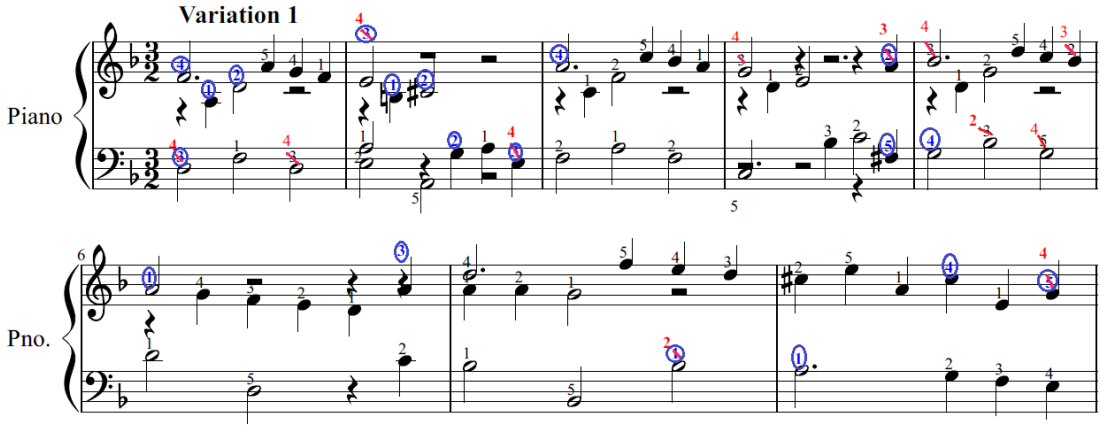


Figure 14: Output for the first eight bars of the first variation on the Saraband from Suite in D minor (HWV 437) by G.F. Händel.

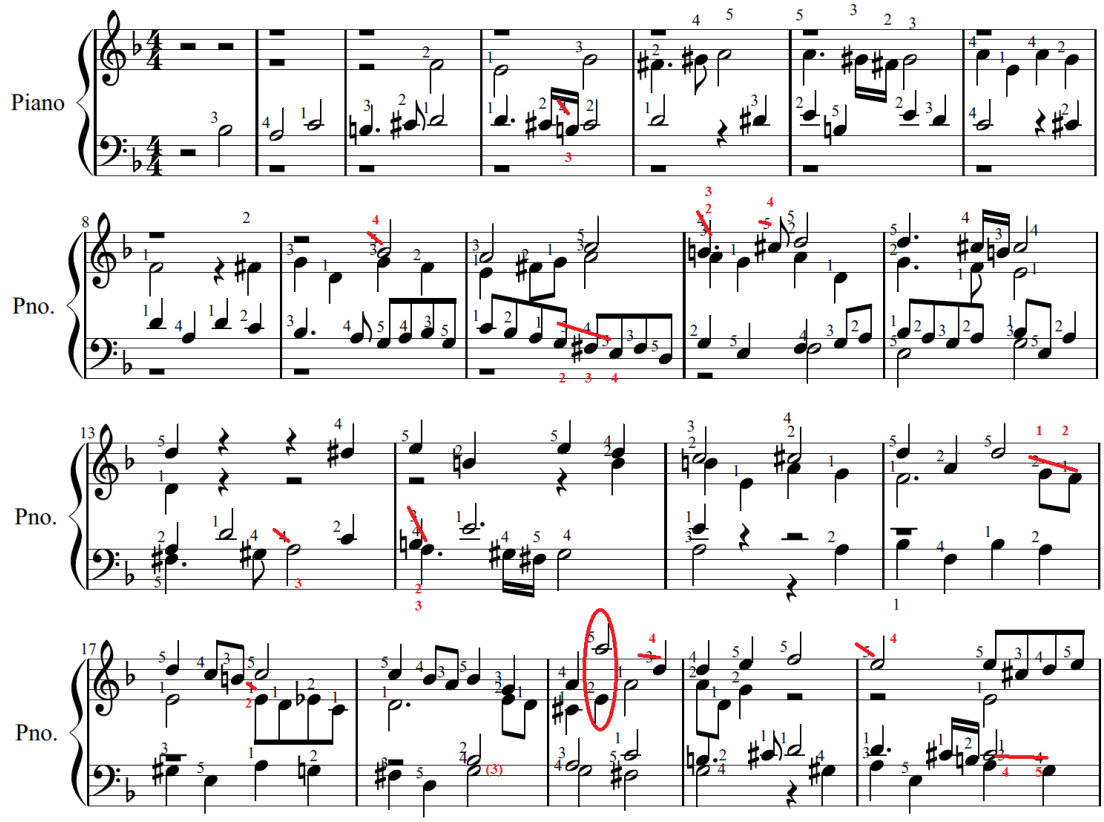


Figure 15: Output for the first 21 bars of the third movement of Contrapunctus XIV from Die Kunst der Fuge (BWV 1080) by J.S. Bach.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

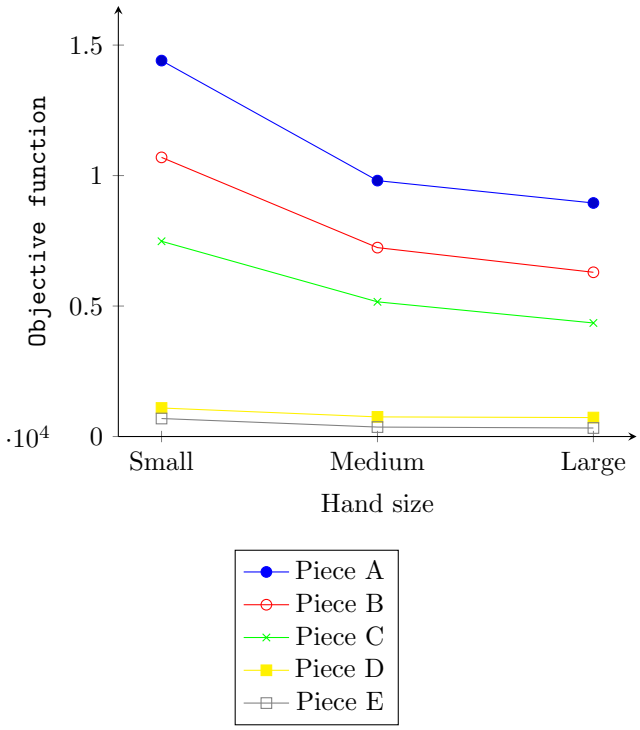


Figure 16: Mean plot for the difficulty of a fingering for each piece using different hand sizes.

to generate fingerings for 5 different pieces. For each piece, the three different hand types were tested separately. For each combination of piece and hand type, 10 outputs were generated. This resulted in a total of $(5 \cdot 3 \cdot 10 =)150$ observations. Three Wilcoxon signed-rank tests were used to test if significant differences exist between the mean value for the `Objective function` for each hand type. Since the objective function is a penalisation for hard to play sequences of notes with a certain fingering, it can be seen as a measure of difficulty of playing. The results of these tests showed that the difficulty of the fingering generated with large hands is significantly lower than the one for medium hands ($p < 0.0001$) and the one for small hands ($p < 0.0001$). In addition, the difficulty for medium hands is lower than the difficulty for small hands ($p < 0.0001$). Figure 16 graphically confirms these findings by showing the average difficulty for each hand size.

The above conclusions are to be expected. Given maximal stretch of a certain finger pair for a large hand will incur a higher difficulty score when using a small hand, because the maximal allowed distance between these fingers will be smaller. As an example, in the second bar of Händels Saraband in Figure 14, the distance between A2 and G3 is fingered respectively with finger 5 and 2 in a large hand. For a smaller hand, the algorithm gives the fingering 5 and 1 as output, because fingers 5 and 2 would be less suited for a small hand to overcome this distance on the piano. This results in worse scores for smaller hands.

7 Conclusions and future research

A good piano fingering is important for a pianist in order to play a piece fluently. We expanded the problem of finding a good piano fingering in order to deal with complete pieces of complex polyphonic music. By basing our evaluation metric on user-specific finger distances, personally tailored solutions can be obtained. In this paper, we developed an efficient VNS algorithm to solve the piano fingering problem in a short amount of time. Whenever a local optimum is reached, a perturbation is performed on the best solution which allows the algorithm to continue the search. Two full factorial experiments were performed to find the optimal setting for each of the

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

components of the VNS. We also showed that all the introduced neighbourhoods have a significant added value. An analysis of the output confirmed the efficiency of the algorithm and its ability to find a good, musically meaningful solution which requires only minimal manual adaptations. The quality of the solution might further be improved by allowing the VNS to run for a much longer time, or by future improvements to the algorithm on one hand and to the objective function on the other hand.

The algorithm might become more powerful if extra neighbourhoods are added. If groups of required manual adaptations show similar characteristics, they might be converted into a move type, as long as the local modification remains small enough to guarantee a limited runtime and high efficiency. Additionally, a mathematical formulation of the problem could lead to the implementation of an exact algorithm, which could be used to find the best possible solution. It would be interesting to compare the performance and output quality of both approaches.

The algorithm currently suggests the crossing of hands when needed, given that the score correctly indicates which notes are to be played with which hand[7]. This is, however, not always the case in existing sheet music, as can be seen in Figure 15. It would therefore be interesting to optimise the choice of hands for each note in a future version of the algorithm. This might be done by allowing the algorithm to move notes from one hand to another when the distance to the closest note(s) in the other hand is below a certain value and when the other hand is not yet fully used, i.e. less than five notes are played simultaneously. Other limitations of the algorithm include special cases such as grace notes and time signature changes.

Another possible improvement could be attained by taking musical sentences into account when new solutions are calculated. Our VNS algorithm reads a whole piece at once, calculating one objective function for each hand in the entire piece. The execution time could be reduced by splitting up a piece into smaller parts for which an individual objective function is calculated, thus reducing the number of possible solutions and neighbourhood sizes. Such a split could be made when a rest is encountered or between two arches, indicating musical sentences.

Another way to make the algorithm work faster is by replacing the generation of a random initial solution by a greedy heuristic that takes into account the relative pitch of the notes within a slice and over the subsequent slices. These could be matched as much as possible to the relative order of the used fingers in the fragment, without compromising the feasibility of the solution. The impact of the initial solution generation could be quantified through another computational experiment.

To obtain the best possible solution, the *set of rules* composing the objective function must be as accurate as possible to reflect the true difficulty of a solution. An alternative approach to deal with this somewhat arbitrary score selection could be to learn the weights based on a corpus of existing piano fingerings.

It might also be useful to implement extra rules from two categories. Firstly, *interpretational rules* are considered. An example of this type of rule is one that favours the use of a strong finger (e.g., the thumb) on the first beat of a bar or a musical sentence. In this paper, it was decided not to do this for two reasons. On one hand, it is not a general rule for every genre to have the first beat stressed. A saraband, e.g., is a dance in which the second beat is stronger. On the other hand, these obvious rules are only a small subset of the interpretational rules, which are piece, genre, pianist and composer dependent (Newman, 1982; Bamberger, 1976). In order to draw good conclusions about a trade-off between interpretation and easiness of a fingering, additional research should be done. One possible approach would be to first define an extensive set of interpretational rules. Then, the impact of each rule might be learned based on specific performers and styles. A second type of rules are those that enhance the *memorisation* of a piece. Such a rule could for instance favour identical fingering patterns for similar note sequences.

Another possible extension of our research would be to use the fingering to determine the difficulty level of a piece. This could be used by score analysing systems as described by Sébastien et al. (2012). Such an application is related to the experiment in Section 6.3, where it is shown that the objective function is worse when a pianist has smaller hands. This objective function score

[7]Right hand notes in the upper staff, left hand notes in the lower staff.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

might be interpreted as an increased difficulty factor. Hence the use of the presented objective function might be expanded for difficulty evaluation purposes.

References

Al Kasimi, A., Nichols, E. & Raphael, C. (2005). Automatic fingering system (AFS). In *poster presentation at ismir, london.*

Al Kasimi, A., Nichols, E. & Raphael, C. (2007). A simple algorithm for or automatic generation of polyphonic piano fingerings. In *8th international conference on music information retrieval, vienna.*

Bamberger, J. (1976). The musical significance of beethoven's fingerings in the piano sonatas. In *Music forum, iv* (pp. 237–280). Columbia University Press.

Chew, E. (2014). Personal communication.

Clarke, E., Parncutt, R., Raekallio, M. & Sloboda, J. (1997). Talking fingers: an interview study of pianits' views on fingering. *Musicae Scientiae*, *1*(1), 87–108.

Eeman, K. (2014). Personal communication.

Forney Jr, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, *61*(3), 268–278.

Gellrich, M. & Parncutt, R. (1998). Piano technique and fingering in the eighteenth and nineteenth centuries: Bringing a forgotten method back to life. *British Journal of Music Education*, *15*(1), 5–23.

Good, M. (2001). Musicxml: An internet-friendly format for sheet music. In *Xml conference and expo.*

Hart, M., Bosch, R. & Tsai, E. (2000). Finding optimal piano fingerings. *The UMAP Journal*, *2*(21), 167–177.

Herremans, D. & Sörensen, K. (2012). Composing first species counterpoint with a variable neighbourhood search algorithm. *Journal of Mathematics and the Arts*, *6*(4), 169–189.

Herremans, D. & Sörensen, K. (2013). Composing fifth species counterpoint music with a variable neighborhood search algorithm. *Expert Systems with Applications*, *40*(16), 6427–6437.

IMSLP. (2014). *Petrucci music library.* Retrieved 5 March 2014, from http://imslp.org/wiki/Main_Page

Jacobs, J. P. (2001). Refinements to the ergonomic model for keyboard fingering of parncutt, sloboda, clarke, raekallio, and desain. *Music Perception*, *18*(4), 505–511.

Koster, A. M., Hoesel, S. P. & Kolen, A. W. (1998). The partial constraint satisfaction problem: Facets and lifting theorems. *Operations Research Letters*, *23*(3–5), 89 - 97. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167637798000431 doi: http://dx.doi.org/10.1016/S0167-6377(98)00043-1

Koster, A. M., Hoesel, S. P. & Kolen, A. W. (1999). Optimal solutions for frequency assignment problems via tree decomposition. In P. Widmayer, G. Neyer & S. Eidenbenz (Eds.), *Graph-theoretic concepts in computer science* (Vol. 1665, p. 338-350). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/3-540-46784-X_32 doi: 10.1007/3-540-46784-X_32

Leijnse, J. (2014). Personal communication.

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

Lin, C.-C. & Liu, D. S.-M. (2006). An intelligent virtual piano tutor. In *Proceedings of the 2006 acm international conference on virtual reality continuum and its applications* (pp. 353–356).

Lourenço, H. R., Martin, O. C. & Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics* (pp. 320–353).

Mladenović, N. & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100.

Nakamura, E., Ono, N. & Sagayama, S. (2014). Merged-output hmm for piano fingering of both hands. In *Proceedings of the 15th international society for music information retrieval conference* (pp. 531–536).

Newman, W. S. (1982). Beethoven's fingerings as interpretive clues. *Journal of Musicology*, *1*(2), 171–197.

Palhazi Cuervo, D., Goos, P., Sörensen, K. & Arráiz, E. (2014). An iterated local search algorithm for the vehicle routing problem with backhauls. *European Journal of Operational Research*, *237*(2), 454–464.

Parncutt, R. (1997). Modeling piano performance: Physics and cognition of a virtual pianist. In *Proceedings of int. computer music conference (thessalonki/gk, 1997)* (pp. 15–18).

Parncutt, R., Sloboda, J. A. & Clarke, E. F. (1999). Interdependence of right and left hands in sight-read, written, and rehearsed fingerings of parallel melodic piano music. *Australian Journal of Psychology*, *51*(3), 204–210.

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.

Radicioni, D. P., Anselma, L. & Lombardo, V. (2004). An algorithm to compute fingering for string instruments. In *Proceedings of the National Congress of the Associazione Italiana di Scienze Cognitive, Ivrea, Italy.*

Robine, M. (2009). Analyse automatique du doigté au piano. In *Proceedings of the journées d'informatique musicale* (pp. 106–112).

Sébastien, V., Ralambondrainy, H., Sébastien, O. & Conruyt, N. (2012). Score analyzer: Automatically determining scores difficulty level for instrumental e-learning. In *Proceedings of the 13th international society for music information retrieval conference* (pp. 571–576).

Sloboda, J. A., Clarke, E. F., Parncutt, R. & Raekallio, M. (1998). Determinants of finger choice in piano sight-reading. *Journal of experimental psychology: Human perception and performance*, *24*(1), 185-203.

Sörensen, K. & Glover, F. (2012). Metaheuristics. In S. I. Glass & M. C. Fu (Eds.), *Encyclopedia of operations research and management science.* New York: Springer.

Swerts, P. (2014). Personal communication.

Yonebayashi, Y., Kameoka, H. & Sagayama, S. (2007a). Automatic decision of piano fingering based on a hidden markov models. In *IJCAI* (pp. 2915–2921).

Yonebayashi, Y., Kameoka, H. & Sagayama, S. (2007b). *Overview of our IJCAI-07 presentation on "automatic decision of piano fingering based on hidden markov models".* Retrieved 18 April 2014, from http://hil.t.u-tokyo.ac.jp/research/introduction/PianoFingering/Yonebayashi2007IJCAI-article/index.html

A Extra tables

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.

Table 9: Example distance matrix for a small right hand (adapted from Parncutt et al. (1997)).

Finger pair	`MinPrac`	`MinComf`	`MinRel`	`MaxRel`	`MaxComf`	`MaxPrac`
1-2	-7	-5	1	3	8	10
1-3	-6	-4	3	6	10	12
1-4	-4	-2	5	8	11	13
1-5	-2	0	7	10	12	14
2-3	1	1	1	2	4	6
2-4	1	1	3	4	6	8
2-5	2	2	5	6	8	10
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 10: Example distance matrix for a medium right hand (adapted from Parncutt et al. (1997)).

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

Preprint accepted for publication in International Transactions of Operations Research. Special Issue on Variable Neighbourhood Search, Published online on 2 October 2015, DOI: 10.1111/itor.12211.