Academic year 2013-2014

UNIVERSITY OF ANTWERP
FACULTY OF APPLIED ECONOMICS

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

Matteo Balliauw
20091027

Master's Thesis submitted to obtain the degree of:

Master of Applied Economic Sciences:
Business Engineering

Promoter:
drs. Dorien Herremans

Co-promoter:
drs. Daniel Palhazi Cuervo

Academic year 2013-2014

UNIVERSITY OF ANTWERP
FACULTY OF APPLIED ECONOMICS

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

Matteo Balliauw
20091027

Master's Thesis submitted to obtain the degree of:

Master of Applied Economic Sciences:
Business Engineering

Promoter:
drs. Dorien Herremans

Co-promoter:
drs. Daniel Palhazi Cuervo

Acknowledgements

I want to thank my promoters, drs. Dorien Herremans and drs. Daniel Palhazi Cuervo for their wonderful assistance in the process of composing this thesis. It has been interesting to learn from them how to program an algorithm, how to conduct experiments with it and how to report this research in a scientific article using LaTeX. Also prof. dr. Kenneth Sörensen has to be thanked for introducing me to the interesting topic of metaheuristics through excellent courses and feedback for this research. Another word of thanks goes to my (future) colleagues at the University and fellow students, that gave me important insights in academic research and a great environment to work and live in. Finally, I thank my parents, grandparents and friends, for supporting me unconditionally and in every way during all the projects that I want to bring to a good end in my life.

Dutch synthesis

Inleiding

Een *pianovingerzetting* duidt aan welke vingers gebruikt moeten worden om de noten op een pianopartituur te spelen. In feite kan elke vinger elke noot op een piano spelen, maar door de specifieke combinatie van noten in een stuk, zijn sommige vingers meer aangewezen dan andere om de noten in kwestie te spelen. Dit in tegenstelling tot een fluit, waar elke noot met een vaste vingerzetting gespeeld wordt (Sloboda et al., 1998). Dit resulteert in een complex probleem, waarbij men rekening moet houden met het niveau van de muzikant, de biomechanica en de gewenste interpretatie. Het beschikken over een goede vingerzetting is bovendien uiterst belangrijk voor een pianist, opdat deze eender welk stuk vlot zou kunnen spelen. Het beslissen over de juiste vingerzetting kan echter een complex en tijdrovend proces vormen (Hart et al., 2000). Automatisatie van dit proces kan in het voordeel zijn van alle pianisten, maar voornamelijk van onervaren amateurs op zoek naar een goede vingerzetting om de online gevonden muziek te kunnen spelen (Yonebayashi et al., 2007a). Bovendien kan dit tijdrovende proces versneld worden voor ervaren muzikanten en kunnen zelfs alternatieve vingerzettingen voorgesteld worden aan professionals (Gellrich & Parncutt, 1998; Parncutt et al., 1999; Sloboda et al., 1998).

Het genereren van een pianovingerzetting kan gezien worden als een *combinatorisch optimalisatieprobleem*. De beslissingsvariabelen zijn discreet, er is een te optimaliseren doelfunctie en de speelbaarheid moet gegarandeerd worden (Neumann & Witt, 2010). Dit probleem kan opgelost worden met exacte oplossingsmethoden, maar ook via metaheuristieken. In deze paper stellen we een efficiënt variabel neighbourhood search algoritme voor dat in een aanneembare rekentijd een zeer goede vingerzetting kan vinden voor zowel de rechter- als de linkerhand en dit in polyfone muziek. Dit algoritme en de parameter instellingen worden grondig getest in een aantal experimenten.

De structuur van deze paper is als volgt: Sectie 2 geeft een overzicht van de huidige stand van zaken van het onderzoeksdomein. Het combinatorisch optimalisatieprobleem wordt uitgewerkt in Sectie 3: de beslissingsvariabelen en de beperkingen worden besproken en de moeilijkheid van een vingerzettings wordt gekwantificeerd. Het nieuwe algoritme wordt voorgesteld in Sectie 4 en de volgende sectie beschrijft de implementatiedetails. In Sectie 6 worden experimenten uitgevoerd om de toegevoegde waarde van de verschillende delen en de optimale parameters van dit algoritme te bepalen. Sectie 7 bespreekt een aantal outputs en enkele aanbevelingen voor toekomstig onderzoek. De belangrijkste conclusies worden gegeven in Sectie 8.

Literatuurstudie

Kwaliteit van de vingerzetting. Bij het opstellen van een vingerzetting moet een afweging gemaakt worden tussen de moeilijkheid van de vingerzetting, het gemak waarmee het stuk gememoriseerd kan worden en de interpretatie van het stuk (Clarke et al. 1997; Sloboda et al. 1998, p. 185). Een ideale vingerzetting bestaat niet, aangezien al deze elementen in rekening gebracht moeten worden (Parncutt et al., 1997). Doorheen de geschiedenis zijn *regels* ontstaan voor het kiezen van een goede vingerzetting. Deze werden zelfs repertoriumafhankelijk (Clarke et al., 1997; Gellrich & Parncutt, 1998). Zoals Robine (2009) aantoont, wordt de interpretatie belangrijker bij professionele muzikanten. Deze wordt ook beïnvloed door de vingerzetting. Daarom worden soms andere vingerzettingen gekozen voor dezelfde stukken, getuigen hiervan Parncutt et al. (1997, p. 369-372). Bijgevolg moet een algoritme de biomechanica van een pianist in rekening brengen. Desalniettemin zijn er een aantal details van de oplossing die onderhevig zijn aan persoonlijke voorkeuren en die niet gemodelleerd kunnen worden. Hiervoor zijn achteraf handmatige aanpassingen vereist (Clarke et al., 1997, p. 100).

De kwantificering van de kwaliteit van een vingerzetting is een cruciaal aspect van het vingerzettingsprobleem. Enkele voorbeelden hiervan zijn in de literatuur terug te vinden. Zo introduceren Parncutt et al. (1997) een kostenminimaliserend paradigma. Hoe hoger deze kost van een vingerzetting, hoe moeilijker het is om een specifiek stuk met de bepaalde vingerzetting te spelen. De auteurs verhogen daarom de kost van een vingerzetting via een aantal regels die bestraffen voor lastig speelbare combinaties en afstanden. Dit model werd licht aangepast door Jacobs (2001).

Een voordeel van deze benadering is dat de set van regels de identificatie van afwegingen tussen verschillende moeilijkheidsbronnen kan aantonen, in tegenstelling tot algemene methoden op basis van overgangsmatrices (Robine, 2009; Hart et al., 2000; Radicioni et al., 2004; Al Kasimi et al., 2005, 2007), een allesomvattende fitness functie (Viana & de Morais Júnior, 2003) of Hidden Markov modellen (Yonebayashi et al., 2007a). Bij deze laatste vorm worden probabiliteiten toegekend aan overgangen en wordt de vingerzetting met de hoogste probabiliteit gekozen als beste alternatief.

Algoritmes. Niet enkel de wiskundige vertaling van het probleem, ook de gehanteerde algoritmes verschillen. Parncutt et al. (1997), Al Kasimi et al. (2005), Al Kasimi et al. (2007) en Robine (2009) maken gebruik van het Dijkstra algoritme en Hart et al. (2000) en Lin & Liu (2006) van dynamische programmering. Een andere benadering is het gebruik van een op regels gebaseerd expert systeem (Viana & de Morais Júnior, 2003). Yonebayashi et al. (2007a) tot slot maken gebruik van het Viterbi algoritme. In dit onderzoek stellen we een nieuw algoritme voor dat erg krachtig is in het oplossen van combinatorische optimalisatieproblemen met exponentiële complexiteit. Aangezien het aantal mogelijke oplossingen exponentieel stijgt met het aantal noten, resulteren n noten in 5^n mogelijke oplossingen. Een exact algoritme dat alle oplossingen zou genereren en er de beste zou uitkiezen, heeft dan een exponentiële rekentijd. Daarom stellen we een sneller algoritme voor dat een zeer goede oplossing kan genereren in een beperkte rekentijd. Metaheuristieken vormen hiervoor een interessant alternatief.

Metaheuristieken. Metaheurstieken bieden richtlijnen om heuristieken te bouwen die oplossingen genereren die op het vlak van de kwaliteit in de buurt komen van exacte algoritmes. Heuristieken zijn gebaseerd op vuistregels en worden vaak op combinatorische optimalisatieproblemen toegepast, aangezien zij toelaten om zeer goede, toegelaten oplossingen te generen in een rekentijd korter dan die van exacte algoritmes (Sörensen & Glover, 2012; Herremans & Sörensen, 2013). Volgens Sörensen & Glover (2012) bestaan er drie klassen van metaheuristieken: local search, constructieve en populatie-gebaseerde. Viana & de Morais Júnior (2003) gebruiken een populatie-gebaseerd, namelijk genetisch algoritme om vingerzettingen te genereren. *Local Search (LS)* daarentegen maakt kleine veranderingen (*moves*) op een bestaande (speelbare) oplossing (Sörensen & Glover, 2012). Dit proces is overigens analoog aan het maken van kleine aanpassingen op een bestaande vingerzetting, zonder dat hierbij alle mogelijke oplossingen beschouwd worden. Dit is namelijk wat pianisten doen, steunend op heuristieken (Sloboda et al., 1998, p. 185).

Metaheuristieken vormen een veelbelovende techniek om deze optimalisatie uit te voeren omwille van hun kracht en snelheid. De mogelijkheid om polyfone muziek voor beide handen in rekening te brengen, zorgt voor complementariteit met het werk van Parncutt et al. (1997) en Jacobs (2001). Bovendien wordt de arbitraire keuze van bestraffing voor moeilijke vingercombinaties vermeden dankzij de mogelijkheid om een aantal parameters aan te passen aan de muzikant zijn hand en voorkeuren. Andere voordelen van het algoritme zijn de mogelijkheid om polyfonie te kunnen verwerken in beide handen en de mogelijkheid om een onderscheid te maken tussen witte en zwarte toetsen (zie Sectie 3.2). Zoals Sébastien et al. (2012) stellen, werd dit vroeger overgeschoten.

Probleembeschrijving

Om een optimalisatieprobleem volledig te beschrijven, moeten we drie elementen specificeren, met name de beslissingsvariabelen, de doelfunctie en de beperkingen. In deze sectie wordt het pianovingerzettingsprobleem van naderbij bekeken.

Beslissingsvariabelen. Voor elke noot van een stuk moet een vinger gekozen worden om deze noot te spelen. Dit zijn de beslissingsvariabelen. De conventie binnen pianovingerzettingen is om elke vinger binnen het hand te nummeren van één tot vijf. De duim is de eerste vinger en deze nummering loopt door tot de pink, die nummer vijf krijgt. Dit wordt geïllustreerd in Figuur 1.

Doelfunctie. De speelbaarheid en kwaliteit van een vingerzetting worden bepaald door de context en de noten die voor en achter de actieve noten komen (Sloboda et al., 1998). Voor dit combinatorisch optimalisatieprobleem moet dus een set van regels gedefinieerd worden die de doelfunctie vormen die vervolgens geoptimaliseerd kan worden. Naar analogie met Parncutt et al. (1997), Jacobs (2001) en Al Kasimi et al. (2005), gebruiken we een aangepaste kostenfunctie om de moeilijkheid van een vingerzetting te kwantificeren en vervolgens te minimaliseren.

Afstandsmatrix Voor de berekening van de doelfunctie, worden zes verschillende afstanden tussen vingerparen gebruikt. Om deze te definiëren, gebruiken we het input model van Parncutt et al. (1997). `MinRel` en `MaxRel` staan voor de kleinste en grootste relaxte afstanden die duiden op een grote eenvoud van overbrugging. `MinPrac` en `MaxPrac` geven informatie over de afstanden die nog net haalbaar zijn voor bepaalde vingerparen. Deze afstanden zijn pianistafhankelijk en kunnen aangepast worden. De laatste twee afstanden zijn `MinComf` en `MaxComf`. Deze tonen de afstanden die zonder belemmering gespeeld kunnen worden.

Afstanden tussen toetsen moeten ook anders gedefinieerd worden om een oplossing te voorzien voor het probleem dat Jacobs (2001) aangeeft. Deze auteur stelt dat het ontbreken van een zwarte toets tussen *mi* en *fa* enerzijds en tussen *si* en *do* anderzijds geen invloed heeft op de afstanden tussen deze toetsen (vergelijk met de afstand tussen *do* en *re* bijvoorbeeld). Imaginaire toetsen (6 en 14 op Figuur 2) worden ingevoerd op de twee plaatsen waar een zwarte toets ontbreekt. Deze nummering gaat door in het volgende octaaf, zodat de volgende *do* nummer 15 heeft. Dit geeft drie voordelen. Ten eerste kan men de afstanden tussen twee toetsen nu berekenen door de waarden voor de toonhoogten van elkaar af te trekken. Ten tweede kan een modulo 2 berekening het onderscheid tussen witte en zwarte toetsen aangeven. Tot slot kan een modulo 14 berekening opnieuw de gebruikte noot aangeven.

Een voorbeeld van een overzicht van deze zes afstanden voor elk vingerpaar in de rechterhand is gegeven in Tabel 1, als aanpassing van Parncutt et al. (1997). Voor kleinere handen kan het programma tabellen met lagere afstanden voor `MinPrac` en `MaxPrac` voorzien of kunnen gebruikers ze naar believen aanpassen, zoals ook door Al Kasimi et al. (2005, 2007) gedaan wordt. Om de afstanden voor inverse vingerparen te berekenen, moet `Min` en `Max` omgewisseld worden en moeten de waarden met -1 vermenigvuldigd worden. Om deze afstand voor de linker hand te bekomen, moet enkel de volgorde van het vingerpaar bij de gewenste parameter omgekeerd worden. Voor transities tussen dezelfde vingers worden alle afstanden op nul gezet.

Regels Deze afstandenmatrix wordt vervolgens gebruikt voor de berekeningen bij de verschillende regels die de doelfunctie vormen. Deze regels zijn opgelijst in Tabel 2. Deze tabel bevat bovendien een kolom met informatie van het type muziek waarop de regels kunnen toegepast worden, omdat ze niet allemaal op monofone of polyfone fragmenten toegepast kunnen worden.

Regels 1, 2 en 13 houden rekening met de afstanden tussen noten en vingers, zoals uitgewerkt door Parncutt et al. (1997) en Jacobs (2001). Dit kan toegepast worden op monofone en polyfone muziek (voor elk notenpaar). Om relatieve volgordes binnen akkoorden in rekening te brengen, worden deze drie regels nogmaals toegepast *binnen* akkoorden in regel 14. Regels 1 en 2 worden hier dubbel zo streng toegepast omdat binnen een akkoord de relatieve volgorde veel belangrijker is. De volgende twee regels (3 en 4) richten zich op positieveranderingen van de hand, die enkel in noodzakelijke omstandigheden mogen toegelaten worden (Parncutt et al., 1997).

Parncutt et al. (1997) gebruikten de afstanden `MinPrac` en `MaxPrac` in regel 13 als bindende beperkingen. Daardoor konden deze afstanden onder geen beding overschreden worden. Als er geen enkele vingerzetting aan deze beperking voldeed, kon dat (monofoon) stuk dus niet legato gespeeld worden. Wij kiezen er echter voor om deze beperking in de doelfunctie op te nemen met zware bestraffingen, in plaats van als harde beperking. Dit verbetert de toepasbaarheid van de regels van Parncutt et al. (1997) op polyfone muziek en verhindert lege oplossingsverzamelingen.

Regels 5 tot en met 11 verhinderen moeilijke fragmenten in monofone muziek. Aangezien de scores voor regel 8 en 9 gemakkelijk vier of vijf kunnen worden voor één vingerpaar (wat relatief zeer hoog is in vergelijking met één of twee in andere regels), wordt de score van deze regels door

twee gedeeld. Wij vinden dat het gebruik van de vierde vinger niet zonder meer vermeden moet worden en ook getraind moet worden. Daarom stellen we de score van regel 5 in dit onderzoek op nul. Dit houdt enkel regel 6 over die de lastige afwisseling tussen de middenvinger en de ringvinger in willekeurige volgorde vermijdt. Regel 12 verhindert op haar beurt het herhaalde gebruik van een vinger in combinatie met een handpositieverandering. Regel 15 wil zulke veranderingen stimuleren op momenten waar andere slices verschijnen, eerder dan tussen identieke slices. De logica achter deze regels werd afgetoetst met professionele muzikanten.

Op het einde worden de punten van alle regels opgeteld per hand, om de doelfunctie per hand te vormen, zoals aangegeven in Vergelijking 2. Deze moet geminimaliseerd worden om de moeilijkheid van de vingerzetting te minimaliseren.

$$\min f(s) = \sum_i f_i(s)$$

Deze benadering heeft als voordeel dat gebruikers hun eigen score aan de regels kunnen toekennen om bepaalde regels zwaarder of minder te laten doorwegen in het eindresultaat.

Beperkingen. De bindende beperkingen binnen dit probleem moeten waken over de absolute speelbaarheid van een vingerzetting. Daarom is de enige speelbaarheidsbeperking in dit onderzoek de regel dat eenzelfde vinger geen twee verschillende noten tegelijkertijd mag spelen. Bijgevolg resulteert het overschrijden van deze regel in het verwerpen van de oplossing.

Variable neighbourhood search algoritme

Zoals eerder gesteld, vormt local search (LS) één klasse van metaheuristieken. Ze starten van een huidige oplossing die voortdurend aangepast wordt (Sörensen & Glover, 2012). Zulke kleine aanpassingen (*moves*) die dezelfde actie uitvoeren, vormen een *neighbourhood*. Na het bereiken van de beste oplossing in zulk een neigbourhood (*lokaal optimum*), wordt een betere oplossing gezocht in een andere neighbourhood om uit dit lokale optimum te ontsnappen. Dit wordt een *variable neighbourhood search (VNS)* zoekstrategie genoemd (Mladenović & Hansen, 1997). In deze paper ontwikkelen we zulk een VNS strategie die behoort tot de LS metaheurstieken. Bovendien voegen we een tabu lijst in.

Een flowchart met de structuur van het algoritme in Figuur 3 toont dat het algoritme start met een willekeurig gegenereerde initiële oplossing. Hierbij wordt aan elke noot een willekeurige vinger toegekend, waarbij steeds aan de bindende beperkingen voldaan wordt. Vervolgens worden respectievelijk de rechter- en linkerhand geoptimaliseerd via een LS in drie neighbourhoods. Een eerste algemene stap is een Swap, waarbij een eerste grote verbetering beoogd wordt door twee vingers over het hele stuk te wisselen. De eerste neighbourhood is Change1 en kan bij één noot één vinger veranderen. Change2 vormt hier een uitbreiding op voor twee noten die in elkaars buurt liggen. De SwapPart is een kleine versie van de Swap, om vingers van simultane noten te kunnen wisselen zonder beperkingen te overtreden.

Figuur 3 toont ook hoe het algoritme door de neighbourhoods loopt. Merk bovendien op dat de splitsing na Swap aangeeft dat de volgorde tussen Change1 en SwapPart 50-50% random is, omdat de performantie stukafhankelijk is. Tijdens het uitvoeren van LS in een neighbourhood, kan men ook rekening houden met een *tabu lijst* die de laatste move(s) bijhoudt die uitgesloten worden voor de volgende veranderingen. Deze noten zijn *tabu active* en de omvang van de tabu lijst heet *tabu tenure* (Glover & Laguna, 1993). In dit algoritme wordt de tabu tenure uitgedrukt als percentage van het aantal noten. Per hand is er één tabu lijst, die geïnitialiseerd wordt bij de start van het algoritme.

Wanneer geen enkele neighbourhood nog een verbetering geeft, kan men uit een lokaal optimum ontsnappen via een *perturbatie*. Dit verandert een willekeurig deel van bepaalde grootte van de beste oplossing om daarna het VNS algoritme opnieuw te starten (Lourenço et al., 2003). Dit wordt ook weergegeven in de flowchart. Perturbaties vervangen hier willekeurig de oplossing van fragmenten in verschillende delen van het stuk. Dit algoritme laat tot slot toe om een aantal parameters te specificeren. Deze worden onderzocht via experimenten in Sectie 6.

Implementatie

Het algoritme is geprogrammeerd in C++. Eerst worden partituren in MusicXML formaat geparst (Good, 2001) via de open source XML Parser van dr. ir. Frank Vanden Berghen (Vanden Berghen, 2013). De informatie wordt opgeslagen in een vector van 'note'-objecten, waarvan de structuur wordt weergegeven in Figuur 5. Een tweede datastructuur met informatie over slices verzamelt de actieve noten per tijdselement. De rusten worden voorgesteld door nullen. Op die manier kunnen alle berekeningen gemaakt worden. Bovendien maakt het algoritme gebruik van een slimme berekening van het verschil in doelfunctiescore. Door enkel te kijken naar het deeltje waarin veranderingen plaatsvinden, vergt deze berekening minder tijd. De output tot slot is een nieuwe XML-file waaraan de vingerzettingen worden toegevoegd. Een plug-in voor de open source muzieknotatiesoftware MuseScore is ook voorzien.

Experimenten

Zoals besproken in Sectie 4, laat het algoritme toe om een aantal parameters aan te passen om tot optimale performantie te komen. Om deze keuze te baseren op een grondige statistische analyse, werden twee experimenten uitgevoerd. Sectie 6.1 bespreekt het experiment dat de parameters van de intensificatiefase van het algoritme vastlegt. Vervolgens wordt het tweede experiment ter bepaling van de parameters van de gehele VNS besproken in Sectie 6.2. Het is mogelijk om eerst de intensificatiefase en vervolgens de hele VNS af te stellen, waarbij rekening gehouden wordt met de resultaten van het eerste experiment. Het voordeel van deze manier van werken, die geïnspireerd is op de paper van Palhazi Cuervo et al. (2014), is dat men minder tijd nodig heeft om deze experimenten uit te voeren. Beide experimenten werden uitgevoerd op een set van tien stukken.

Intensificatiefase. Het eerste experiment ter bepaling van de VNS, bestudeert zes parameters. De parameter `step_swap` voor de `Swap` stap en `nbh_X` voor elke neighbourhood duidt op activatie (On) of deactivatie (Off) in het algoritme. De parameter `tabusize` duidt de tabu tenure aan als een percentage van het aantal noten in het stuk. De laatste parameter `improv_strat` wijst op de verbeteringsstrategie: beste of eerste verbetering. De geteste waarden worden weergegeven in Tabel 3. Twee afhankelijke variabelen worden bestudeerd. De kwaliteit van de oplossing wordt bekeken via de opgetelde scores van de doelfunctie voor links en rechts, aangeduid als `Objective function`. De variabele `Runtime` geeft dan weer informatie over de totale computerrekentijd om tot deze oplossing te komen. Aangezien alle parametercombinaties tien keer getest werden per stuk en er tien stukken waren, werden er 12800 observaties bekomen.

Deze observaties werden geanalyseerd via een mixed-effects analysis of variance (ANOVA) met de statistische software JMP. Het stuk wordt ingecalculeerd als random effect. De p-waarden horende bij de F-testen, worden weergegeven in Tabel 4. Hierin zit ook een selectie van interactie-effecten. Deze tabel toont aan dat het invoeren van elke stap of neighbourhood een significante impact heeft op de doeltreffendheid van het algoritme. Figuur 6 toont dat deze impact telkens positief is, aangezien een lagere score bereikt wordt. De tabu tenure heeft ook een significante impact op de score, maar deze blijkt negatief te zijn.

Ook voor het model voor de variabele `Runtime` zijn de p-waarden opnieuw terug te vinden in Tabel 4. De bijhorend gemiddelden zijn grafisch weergegeven in Figuur 6.

Het is wel opmerkelijk dat de gehanteerde verbeteringsstrategie geen invloed blijkt te hebben op de score, noch op de rekentijd. Als we de interactie-effecten met neighbourhoods bestuderen, zien we dat er wel impact is op de rekentijd, maar dat deze afhankelijk is van de neighbourhood (zie Tabel 4 en Figuur 7). Een grafisch voorbeeld in Figuur 8 toont de evolutie van de score over tijd, eenmaal met de beste verbetering, andermaal met de eerste verbetering. Het blijkt dat eerste verbetering meer stapjes nodig heeft, aangezien dit kleinere verbeteringen maakt in vergelijking met beste verbetering. Deze kleine stapjes nemen echter wel minder tijd in, aangezien dan niet de hele neighbourhood doorzocht moet worden. Die negatieve correlatie tussen rekentijd en hoeveelheid verbetering zorgt er voor dat beide strategieën een gelijke rekentijd nodig hebben. Bovendien is ons algoritme robuust, wat zorgt dat ook de finale score niet significant beïnvloed wordt.

Uit de analyse hier gemaakt, besluiten we om elke stap en neighbourhood in het algoritme op te nemen en de tabu lijst te deactiveren. We kiezen bovendien voor beste verbetering, aangezien het over het algemeen iets beter en sneller blijkt te zijn. Deze uiteindelijke waarden worden weergegeven in Tabel 5.

Gehele VNS. Na het bepalen van de parameters voor de intensificatie fase in het vorige experiment, worden hier de beste waarden voor de parameters van de gehele VNS strategie vastgesteld. Er zijn drie parameters. Het aantal herhalingen zonder verbetering wordt weergegeven door `nrofiterations`. Het relatief aantal noten dat willekeurig veranderd wordt, duiden we aan met `sizeofpert`. De parameter `partsize` geeft de grootte van de delen waarin geperturbeerd wordt weer als percentage van het aantal slices. Delen van 10% resulteren dus in tien delen zoals beschreven in Sectie 4. De geteste waarden worden weergegeven in Tabel 6. De bestudeerde afhankelijke variabelen zijn dezelfde als in het vorige experiment. De bestudeerde range werd vastgesteld op basis van een eerste kleine test. In totaal werden hier 6400 waarnemingen bekomen.

De mixed-effects ANOVA hier neemt opnieuw tweedegraads interactie-effecten in rekening om de impact van een combinatie van parameters op de performantie van de perturbatie te bestuderen. Tabel 7 toont dat de drie parameters een significante impact hebben op beide afhankelijke variabelen. De invloed van de grootte van de perturbatiedelen wordt weergegeven in Figuur 9. De beste instelling blijkt stukken van 25%, of 4 delen per stuk te zijn. Voor de parameters `nrofiterations` en `sizeofpert` is het logisch dat een toename van de waarden hiervan effectief resulteert in betere prestaties van het algoritme en een langere rekentijd. Tabel 7 toont bovendien dat het interactie-effect een significante impact heeft op beide afhankelijke variabelen. Deze worden in Figuur 10 grafisch voorgesteld. Hieruit blijkt dat de marginaal toegevoegde waarde van een toename van één van deze parameters afnemend is als de andere parameter hoog is. Bovendien neemt de benodigde rekentijd sterk toe. Daarom is het verder verhogen van `nrofiterations` boven tien en `sizeofpert` boven 20% van weinig toegevoegde waarde. Enkel de rekentijd zou omhoog gaan. De beste waarden die uit deze analyse volgen, staan opgelijst in Tabel 8.

Resultaten en aanbevelingen voor verder onderzoek

Resultaten. Na het bepalen van de beste parameters voor het algoritme, kunnen de prestaties van het programma bestudeerd worden op een nieuw stuk. Hiervoor wordt de monofone vierde Menuet van J. S. Bach (BWV Anh. 114) gekozen. Het duurde 110 seconden om tot de uiteindelijke oplossing te komen. De evolutie van de score over tijd voor de rechter hand is weergegeven in Figuur 11. Het toont dat de perturbatie strategie een efficiënte oplossing is om uit een lokaal optimum te ontsnappen.

De output van de eerste acht maten in Figuur 12 toont welke vingerzettingen acceptabel zijn of niet. De dikke cijfers duiden op de vingerzettingen die noodzakelijkerwijze aangeduid worden opdat de pianist alle andere ook zou kunnen afleiden. Hiervan is er slechts één fout, in vergelijking met de originelen in het muziekboek. Enkele andere fragmenten vereisen ook kleine manuele aanpassingen.

Een soortgelijke analyse is ook mogelijk voor een polyfoon stuk, namelijk de eerste variatie van G. F. Händels Sarabande uit de Suite in Re klein (HWV 437). De uiteindelijke oplossing werd bekomen na 38 seconden. De evolutie van de score over de tijd wordt voor de rechter hand weergegeven in Figuur 13. Een deel van de output staat in Figuur 14. Vergeleken met de oorspronkelijke vingerzettingen van het boek, zijn er enkele die kleine aanpassingen vereisen.

Verder onderzoek. Om deze manuele aanpassingen te vermijden, kan men het algoritme toelaten om veel langer te lopen. Ook andere aanpassingen aan het algoritme of de doelfunctie kunnen voor verbeteringen zorgen.

Verbeteringen aan het algoritme Het algoritme kan in eerste instantie krachtiger worden door meer neighbourhoods toe te voegen. Deze kunnen geïnspireerd zijn op types van manuele aanpassingen. Vervolgens zou het algoritme ook sneller kunnen werken door het stuk op te splitsen in aparte delen bij rusten of het einde van muzikale zinnen. Dit kan als voordeel hebben dat

muzikale zinnen beter weerspiegeld worden in de gekozen vingerzettingen. Een andere versnelling kan bereikt worden door de willekeurige initiële oplossing te vervangen door een *greedy* heuristiek die de relatieve toonhoogte en gekozen vingerzetting zoveel mogelijk probeert in overeenstemming te brengen. De impact hiervan kan bekeken worden via een experiment zoals deze uit Sectie 6 en zoals deze van Palhazi Cuervo et al. (2014).

Verbeteringen aan de doelfunctie Om tot een zo goed mogelijke oplossing te komen, moet de set van regels zo accuraat mogelijk zijn. Regel 6 bijvoorbeeld verhindert het na elkaar gebruiken van vinger 3 en 4. Dit kan leiden tot geforceerde en onlogische oplossingen. Daarom adviseren we gebruikers om de score van deze regel zeer laag te maken, zodat enkel bij indifferentie tussen twee alternatieven dit gebruik vermeden wordt, of zelfs voor score nul, zodat constructies als deze in maat 6 van Figuur 12 vermeden worden. Ook parameter training zou kunnen bijdragen aan de accuraatheid van de regels.

Men zou ook kunnen stellen dat interpretationele regels toegevoegd moeten worden. Een voorbeeld hiervan is het gebruik van een sterke vinger (bv. de duim) op de eerste tel van een maat. Dit werd hier niet gedaan om twee redenen. Enerzijds is het geen algemene regel. In een sarabande bijvoorbeeld wordt namelijk de tweede tel benadrukt. Daarenboven vormen deze duidelijke regels slechts een beperkte subset van de interpretationele regels, die op hun beurt stuk, pianist en componist afhankelijk zijn (Newman, 1982; Bamberger, 1976). Om goede conclusies te kunnen trekken over de afweging tussen interpretatie en eenvoud van een vingerzetting, is aanvullend onderzoek nodig. Men moet eerst een zo volledig mogelijke set van interpretationele regels vaststellen. Daarnaast zou ook hier parameter training nuttig kunnen zijn.

Uitbreiding van het onderzoek Een laatste uitbreiding van dit onderzoek is de kwantificering van de moeilijkheid van een stuk en haar vingerzettingen via een parameter die de score van de doelfunctie en de lengte van het stuk in rekening brengt (bv. berekend via het aantal noten en het aantal maten). Dit zou gebruikt kunnen worden in analyse software zoals beschreven door Sébastien et al. (2012).

Conclusies

Een pianist heeft een goede vingerzetting nodig om een stuk vlot te kunnen uitvoeren. Op basis van de literatuur werd de probleembeschrijving uitgewerkt en uitgebreid om hele stukken polyfone muziek voor beide handen te kunnen verwerken. Door de doelfunctie uit te breiden en het aantal dwingende beperkingen te reduceren, werd een realistischere doelfunctie bekomen die ook rekening kan houden met het verschil tussen witte en zwarte pianotoetsen. Via gebruikersafhankelijke inputs, worden aangepaste oplossingen mogelijk.

Het voorgestelde VNS algoritme is een LS metaheuristiek, waarbij na een lokaal optimum een perturbatie uitgevoerd wordt op de beste oplossing en het algoritme herstart wordt. Twee experimenten lieten ons toe om de beste parameters voor het algoritme vast te stellen.

Een daaropvolgende test toonde de toegevoegde waarde van de onderdelen van de metaheuristiek en de efficiëntie van het algoritme aan in de praktijk. Enkele suggesties te verbetering van het algoritme en de doelfunctie werden hierbij ook besproken. Andere aanbevelingen voor verder onderzoek zijn ook op deze resultaten gebaseerd. Het zou interessant kunnen zijn om te kijken hoe het toevoegen van interpretationele regels het algoritme beïnvloedt. Tot slot kan ook getracht worden om de moeilijkheid van een stuk aan de hand van de doelfunctie te kwantificeren.

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

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

ANT/OR, University of Antwerp Operations Research Group, Prinsstraat 13, 2000 Antwerp, Belgium

Abstract

A good piano fingering is essential for pianists to play a piece fluently. Some examples of algorithms to generate a piano fingering automatically can be found in the literature. However, the existing methods can only generate fingerings for small, monophonic piano pieces for the right hand, minimising the difficulty. In this paper, we develop a variable neighbourhood search (VNS) algorithm that can generate good fingerings for large pieces of polyphonic piano music considering both hands in an acceptable amount of calculation time. The algorithm calculates the difficulty based on a user-specific finger distance matrix. The set of rules considered in the literature in order to define the quality of a fingering is expanded to deal with specific finger combinations in polyphonic music. The VNS is a local search metaheuristic that searches in different neighbourhoods and performs a perturbation when a local optimum is reached. The selection of the optimal parameters of the VNS and the perturbation strategy are determined using thorough computational experiments and statistical analysis.

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

1. Introduction

A *piano fingering* is an indication of the fingers that should be used to play the notes that compose a piano piece. In principle, any finger could play any note on a piano, as opposed to flute or recorder fingering with fixed finger positions for every note. Depending on the previous and the following notes in a piano piece however, some fingering combinations are easier or better suited than others (Sloboda et al., 1998). As a result, the choice of a good piano fingering, considering the pianists level, his or her biomechanics and the composers desired interpretation, allows a pianist to play a piece fluently, even the more difficult fragments. However, deciding on the right fingering for a piece of music can be a complex, time consuming and burdensome task (Hart et al., 2000). As a result, automation of the process could help all pianists, but especially inexperienced hobbyists to find good fingerings, enabling them to play pieces more easily (Yonebayashi et al., 2007a). The large amount of digital sheet music, transcriptions or own compositions available on-line these days are a treasure for any pianist. Unfortunately, in many cases they don't have any fingering. This shows the need of having an automatic tool to calculate the fingering. Even for more experienced pianists however, automation of the process could reduce the time needed for getting their preferred fingering, thus giving them more time to actually study the piece. It might even suggest alternative possibilities to professionals (Gellrich & Parncutt, 1998; Parncutt et al., 1999; Sloboda et al., 1998).

Generating a piano fingering can be seen as a combinatorial optimisation problem. The decision variables are discrete, there is an objective function that needs to be optimised and the feasibility

*Corresponding author

Email address: matteo.balliauw@uantwerpen.be (Matteo Balliauw)

of the solution has to be guaranteed (Neumann & Witt, 2010). These kind of problems can be solved by exact solution methods, but also by metaheuristics. In this paper, we propose an efficient variable neighbourhood search algorithm that solves the fingering problem for polyphonic music in both hands in a reasonable amount of computational time. This algorithm and its parameter settings are thoroughly tested in a computational experiment.

The structure of this paper is as follows: Section 2 gives an overview of solutions to this problem proposed by other researchers. The combinatorial optimisation problem is defined and elaborated in Section 3: we discuss the decision variables, we quantify the difficulty of a fingering and describe the constraints. Here, the existing rules for evaluating the fingering quality are expanded, including a user-specific finger distance matrix. In Section 4, the developed VNS is discussed and in Section 5, the implementation details are described. Section 6 elaborates on experiments that identify the components with a positive impact on the optimisation process and the optimal values for the parameters of the algorithm. Section 7 shows some outputs of the algorithm and gives recommendations for further research. The major conclusions are shown in Section 8.

2. Literature review

2.1. Measuring the quality of a fingering

When deciding on a fingering, a trade-off between difficulty of playing, easiness of memorisation and interpretation should be made (Clarke et al. 1997; Sloboda et al. 1998, p. 185). One ideal fingering does not exist, as one has to take these three objectives into account (Parncutt et al., 1997). Firstly, the *difficulty of playing* is related to the size and biomechanics of the hand. Secondly, some musicians can *memorise* certain combinations more easily than their colleagues. Thirdly, *personal preferences* for certain fingering combinations exist. This is reinforced by the difference between pianists in expressing musical-technical elements or feelings in a piece, sometimes suggested by the composer. Throughout the history, different *rules* emerged for deciding on an appropriate piano fingering. They even became repertory dependent (Clarke et al., 1997; Gellrich & Parncutt, 1998). In the case of pianists with more professional experience, the potential to express certain musical interpretations with different fingering combinations becomes more important and less attention is paid to the simplicity of the fingering or the cognitive aspect. Robine (2009) shows that the chosen fingering influences the musical execution. This author gives small lags between two notes, caused by passing the thumb underneath another finger, as a specific example. For these reasons, different pianists choose different fingerings for the same piece, as proven by Parncutt et al. (1997, p. 369-372). Hence, an algorithm that generates good piano fingerings should try to take the personal biomechanics of a pianist into account through rules. However, some details of these fingerings are subject to personal preferences and cannot be modelled, which requires some manual adaptations afterwards (Clarke et al., 1997, p. 100).

The quantification of the *quality of a fingering* is the key aspect to define for the piano fingering problem. One of the first to introduce a cost minimising paradigm to quantify the difficulty of a piano fingering in the literature, was the paper by Parncutt et al. (1997). The higher the cost of a fingering, the more difficult it is to play the considered piece with that specific order of fingers. The authors increase the cost of a fingering, based on rules that penalise for difficult sequences and distances in a fingering. This method is also applied by Parncutt (1997) and Lin & Liu (2006) and expanded by Jacobs (2001). An important feature of this cost minimising paradigm, is that they work with 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, contrary to other methods where only a general model with one origin of difficulty is defined through transition matrices (Robine, 2009; Hart et al., 2000; Radicioni et al., 2004; Al Kasimi et al., 2005, 2007) or through a general fitness function (Viana & de Morais Júnior, 2003). Yonebayashi et al. (2007a) use a different approach without cost functions, namely Hidden Markov Models, where probabilities are assigned to transitions. The fingering with the highest probability of selection by potential pianists, is then considered to be the best alternative.

2.2. Algorithms

The algorithms available in the literature for the piano fingering problem differ widely. Parncutt et al. (1997), Al Kasimi et al. (2005), Al Kasimi et al. (2007), and Robine (2009) e.g. make use of the Dijkstra algorithm and Hart et al. (2000) and Lin & Liu (2006) of dynamic programming. Another approach is the use of a rule based expert system by Viana & de Morais Júnior (2003). Yonebayashi et al. (2007a) use the Viterbi algorithm to find the optimal solution. In this paper, we implement an algorithm that has proven to be very successful at solving other combinatorial optimisation problems. As the number of solutions grows exponentially with the size of the piece (with n notes, there are 5^n possible solutions), an exact algorithm enumerating all possibilities would have an exponential runtime. Therefore, a quicker algorithm that can result in a very good solution should be developed. For this reason, metaheuristics form an interesting alternative.

2.3. Metaheuristics framework

Metaheuristics offer guidelines to build high quality heuristic algorithms. Heuristics are based on rules of thumb and they are often applied in order to solve combinatorial optimisation problems, as they can result in a very good and feasible solution in an execution time that is shorter than that of exact algorithms (Sörensen & Glover, 2012; Herremans & Sörensen, 2013). According to Sörensen & Glover (2012), there exist three types of metaheuristics: local search, constructive and population-based. Viana & de Morais Júnior (2003) implement a population-based metaheuristic, called genetic algorithm, in the domain of finding fingerings for an instrument. Local search in turn, makes small adaptations (*moves*) to an existing (feasible) solution (Sörensen & Glover, 2012). The process of making small changes to a fingering is analogous to an artist implementing some final adaptations to a given fingering (whether or not indicated by the composer), without considering all solutions, as Sloboda et al. (1998, p. 185) describe the human process that requires some heuristics.

Because metaheuristics can generate good-quality solutions in a reasonable execution time, this is a promising optimisation framework to calculate good fingerings for both hands in polyphonic pieces. In this way, this research can be complementary to the work of Parncutt et al. (1997), Jacobs (2001) and Al Kasimi et al. (2005). Moreover, the arbitrary selection of penalties for difficult fingering combinations is tackled through the ability of setting some parameters according to the musicians hands and preferences. In order to do so, a tailor made version of the rules of Parncutt et al. (1997) is elaborated. Additional benefits of the algorithm are that it can deal with polyphony, the use of the left hand and the difference between white and black keys, as discussed in Section 3.2. These cases were previously ignored, as Sébastien et al. (2012, p. 574) indicated.

3. Problem description

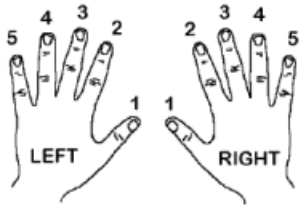
In order to define an optimisation problem unambiguously, three elements have to be specified: the decision variables, the objective function and the constraints. Here, the problem of finding an optimal fingering is presented as a *combinatorial optimisation problem*. For every note, a finger that plays that note has to be chosen, composing the piano fingering. The solution space is discrete, as one of the five fingers of either the right or the left hand should play a note[1]. The generation of a piano fingering is rule based, as the quality of a fingering is expressed through a set of rules that penalises for difficult combinations. As a result, the goal is to find the fingering with the lowest penalty score. Finally, feasibility constraints ensure the playability of fingering combinations (Neumann & Witt, 2010).

[1]In piano music, the convention is made that the upper staff is played by the right hand and the lower staff is played by the left hand.

3.1. Decision variables

For every note of a piece, a finger has to be chosen to play that note. The choice of the fingers are the decision variables. The generally accepted convention in fingering is to code every finger of the hand with a number from one up to five. The first finger is the thumb and the numeration goes up to the little finger, which receives number five. This is shown in Figure 1.

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



3.2. Objective function

Feasible and good piano fingering positions depend on the context of notes preceding and following current notes (Sloboda et al., 1998). For this combinatorial optimisation problem, the intrinsic set of rules to define the quality of a fingering is quantified in an objective function. In analogy with Parncutt et al. (1997), Jacobs (2001) and Al Kasimi et al. (2005), an adapted version of a cost or difficulty function that penalises for difficult combinations is minimised. Before explaining the rules however, the representation of allowed distances between fingers used in the objective function calculation, is discussed.

3.2.1. Distance matrix

We use the convention of Parncutt et al. (1997), where six different types of distances affecting the rules in the objective function are used. `MinRel` and `MaxRel` stand for minimal and maximal relaxed distances, which means that these distances can easily be played with the corresponding finger pairs. `MinPrac` and `MaxPrac` give information about which distances per finger pair are the biggest ones possible to play. These distances are user-specific and can be adapted by the user in the distance matrix. The two final distances are `MinComf` and `MaxComf`. These show the distances that can be played without hamper. They depend on the values of `MinPrac` and `MaxPrac`, as can be seen from Equation 1 (Parncutt et al., 1997).

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

Distances between keys are redefined in order to solve the problem issued by Jacobs (2001). Jacobs states that a missing black key between E and F on one hand, and between B and C on the other hand does not influence the distance between these respective note pairs. Imaginary, unused pitches (key 6 and 14 on Figure 2) are introduced in the two positions 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 system has several advantages: firstly, the distance between two keys can be calculated by subtracting the coding values for each pitch. Moreover, by calculating modulo 2, one can distinguish between black and white keys. Finally, by calculating modulo 14, the pitch class can be decoded.

An example of an overview of the distances for each fingering pair of 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 and based on interviews with professional musicians. For smaller hands, tables with reduced absolute values of `MinPrac` and `MaxPrac` are available in the program, or users could

Figure 2: Piano keyboard with additional imaginary black keys.

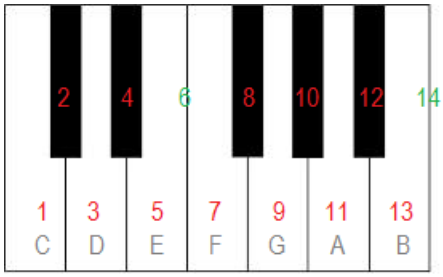


Table 1: Distance matrix for the right hand (adapted from Parncutt et al. (1997)).

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

adapt them according to their preferences, as was also considered by Al Kasimi et al. (2005, 2007). The `Comf` distances are adapted accordingly and the `Rel` distances remain the same.

In order to calculate the distances for the fingering pair where the order of the firstly and secondly used finger 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 in 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 fingering pair just has to be swapped (e.g. 1-2 becomes 2-1). E.g. to calculate `MinPrac_L`(2-1) $=$ `MinPrac_R`(1-2) $= -10$.

All distances for reusing the same finger (e.g. 1-1) are always set to zero.

3.2.2. Rules

The distance matrix is used to calculate how well a fingering adheres to the set of rules. These rules are listed in Table 2, which includes a description, the penalty and the origin of these rules. Moreover, not every rule can be applied on all types of music. The *application* column indicates if the rules are applied on monophonic, polyphonic or both types of music.

Table 2: Set of rules composing the objective function (adaptated 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	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	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 (in any consecutive order).	+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 (in any consecutive 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 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 note, played by the same finger, and the second pitch being the middle one.	+1	Own rule
13	All	For every unit 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 14 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

Rules 1, 2 and 13 take the accepted distances between different notes into account and are based on the work of Parncutt et al. (1997) and Jacobs (2001). These rules can be applied on monophonic and polyphonic music (for each pair of notes). To account for acceptable relative orders of fingers in chords, rules 1, 2 and 13 are also applied *within* a chord in rule 14 to avoid unhandy positions like a right index placed on a lower note than the right thumb within one chord. Rules 1 and 2 are applied in this case with doubled scores to account for the importance of these relative ordering in one chord. Rules 3 and 4 account for hand position changes, which have to be reduced to necessary cases only (Parncutt et al., 1997).

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 cannot be violated. If no fingering is feasible, that (monophonic) part can thus not be played legato[2]. We, however, do not implement these practical distances as hard constraints. They are included as soft constraints in the objective function, resulting in high penalties per unit of violation. This improves the applicability of the set of rules of Parncutt et al. (1997) on polyphonic music and allows us to start from a feasible solution.

Rules 5 up to 11 try to prevent difficult fragments in monophonic music. The original scores of rule 8 and 9 could respectively become 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 here. In addition, the use of the fourth finger should not be avoided at all cost and should be trained. Hence we advise the user to change the score in rule 5 to zero. This only leaves rule 7 preventing the difficult use of finger three and four in any consecutive order.

Rule 12 prevents the repetitive use of a finger in combination with a hand position change. E.g.: a sequence C4-G4-C5 fingered 2-1-2 in the right hand makes a pianist reuse the second finger too quickly. This becomes problematic in fast pieces, so using the combination 2-1-3 would be a better approach. Rule 15 tries to favour hand position changes at the moment when consecutive slices are different. Otherwise, the same notes are sometimes played by a different finger, although they are following one after another. The logic of these rules and their relative scores came from a discussion with professional musicians: piano teacher Kathleen Eeman, dr. Piet Swerts from K.U. Leuven, dr. Elaine Chew from the Queen Mary University of London and biomechanician dr. Joris Leijnse, researching hand anatomy and music at the ULB.

At the end, the points resulting from each rule should be summed for each hand, to form the total score of difficulty per hand, as shown in Equation 2. This has to be minimised in order to minimise the difficulty of the fingering.

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

This approach has an important advantage. It allows the user to change the scores of different rules to make some important rules count more severe or make them less important (e.g. pianists who do mind using the fourth finger, might set the score of rule 5 back to one in stead of zero).

3.3. Feasibility constraints

The constraints of this problem account for the absolute playability of a fingering combination. In this sense, the only feasibility constraint is that the same finger cannot be used on two different active keys played at the same time. A violation of this constraint results in a rejection of the solution[3].

[2]Smoothly connected.

[3]This also excludes the use of one single finger on two adjacent notes (e.g. thumb on two adjacent white keys). The actual algorithm does include the possibility to assign different fingers to a longer note consisting of different tied notes. In this way, finger substitutions are incorporated in a piece.

4. Variable neighbourhood search algorithm

As stated before, one class of metaheuristics are local search (LS) metaheuristics. They start from a current solution, that is continuously adapted. Such small adaptations in a local search metaheuristic (*moves*) doing the same action (e.g. a move type that changes one element of the solution) are part of a *neighbourhood* (Sörensen & Glover, 2012). After reaching the best solution within this neighbourhood (*local optimum*), a better solution is searched in a different neighbourhood to get out of this local optimum. This is called a *variable neighbourhood search (VNS)* strategy (Mladenović & Hansen, 1997). In this paper, we develop such a VNS strategy that belongs to the local search metaheuristics and additionally, we included a tabu list. Also Herremans & Sörensen (2012) applied VNS successfully to a musical problem, namely to automatically compose counterpoint music.

As can be seen from the flowchart diagram of the algorithm in Figure 3, the algorithm starts with the generation of a random initial solution. A random finger is attributed to every note in all the *slices* that compose a piece. A slice is the smallest unit of time into which the piece can be divided (see also Section 5). In order to obtain a feasible solution, once a finger is used in a slice, this finger can no longer be attributed to another note in the same slice. As this process works its way through the piece from left to right and assuming that only five notes can be played at the same time by one hand, a feasible solution is guaranteed. Afterwards, respectively the right and left hand are optimised separately. An initial preprocessing step that aims to make a large first improvement by interchanging all fingers f and g throughout the piece, is called Swap.

Afterwards, LS takes place in three neighbourhoods, each defined by a move type, improving the solution after each move. An example move from every neighbourhood is displayed in Figure 4. The move Change1 changes the fingering of every note separately to any other allowed finger, resulting in a neighbourhood with feasible fragments in which one finger is changed. In the Change2 move, this strategy is expanded to two notes that are both adjacent or simultaneously played, thus resulting in fragments with two changed fingers. Finally a SwapPart move is included to enhance the changeability of the fingering f (to g) of one long note a, during which other notes b are played with finger g. This move changes the fingering of note a to g and the fingering of the notes b, fully contained within the length of note a, to f. Only feasible results are stored in the neighbourhood. In the example, the D5 is note a, played with finger 1. Here, the considered simultaneous notes are F4, A4 and again F4 (the G4's begin before or end after the D5). Of these three notes, both the F4's are the notes b, played with finger 5. These fingers 1 and 5 are interchanged in a move from SwapPart. This move is necessary to be able to change the fingering of the longer note in a similar situation, without obtaining a neighbourhood that is too large and that would no longer qualify as a part of a local search. The fingering of the longer note can only be changed from 1 to 5, when all the simultaneous notes played with 5 are changed to the previous fingering of the long note. Otherwise, simultaneous notes are played with the same finger, resulting in an infeasible solution. We also limit these simultaneous notes to notes fully contained within the interval of the longer note in order to reduce the probability of arriving at an infeasible solution, caused by notes preceding or following the long note a.

Figure 3 also shows how the different neighbourhoods are chained in the VNS strategy. When performing LS in a neighbourhood, the algorithm can take into account a *tabu list*, that keeps a record of the notes included in the last move(s) and that cannot be included in the next move(s). These notes are *tabu active* and the size of this list is denoted as the *tabu tenure* (Glover & Laguna, 1993). In this algorithm, the tabu tenure is expressed as a percentage of the number of notes. A separate tabu list for each hand is initialised at the start of the algorithm, is updated after the execution of a move and remains active until the end of the algorithm is reached.

The algorithm continues to execute LS in one 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 (Change2) is reached, the algorithm goes back to the point after the Swap move, as this move is only used as a first general improvement. The flowchart also shows a split after the Swap step. This indicates that the order of Change1 and SwapPart is chosen randomly (with a 50-50 % probability) by the algorithm. This is because the

Figure 3: Structure of the algorithm.

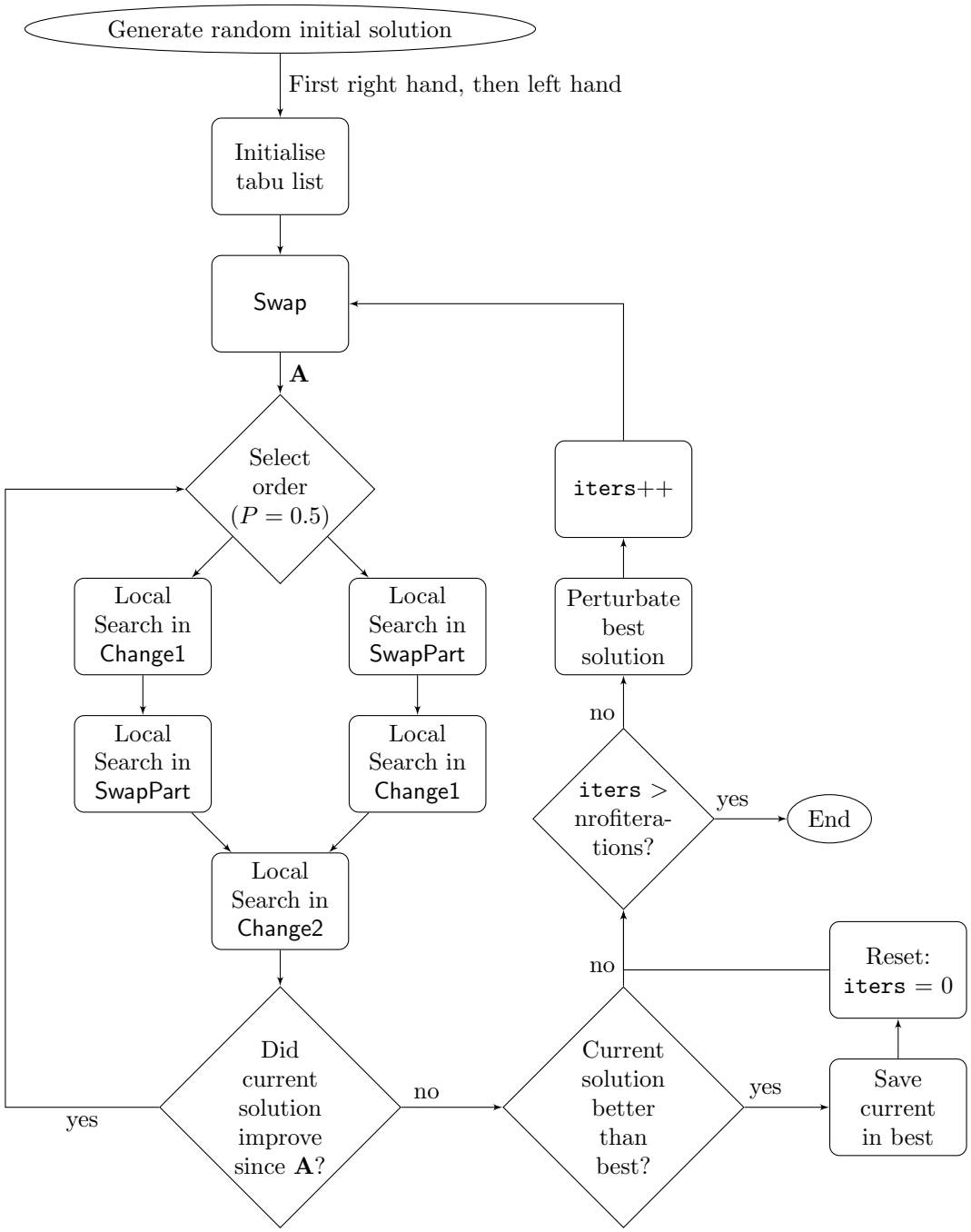
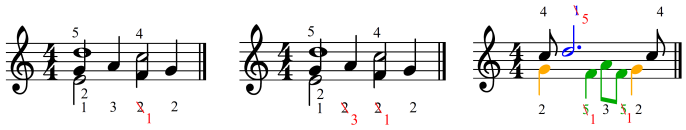


Figure 4: Examples of a move from the different neighbourhoods.



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

best order is fragment dependent as its performance depends on the monophonic or polyphonic character of the fragment.

This search strategy continues up to the point where a loop through the last three neighbourhoods can find no more improvement. 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 is reset to zero. When the maximum number of iterations is reached, the algorithm stops. Otherwise, a *perturbation* is performed on the best solution. A perturbation in general randomly changes a segment of a predefined size of the best solution to restart the VNS afterwards (Lourenço et al., 2003). Here, it removes the fingerings from a random fragment of a certain size in different *parts* of the best solution and randomly assigns new fingers to these notes. Because these new fingers are only chosen from unused fingers in that slice, the feasibility of the solution is guaranteed as with the generation of the random initial solution. The resulting solution is handed back to the Swap step and a following iteration of the algorithm is initialised.

As can be noticed from the previous, the different components of the VNS allow us to specify many parameters. The best values of these parameters are determined through the computational experiments described in Section 6.

5. Implementation

The algorithm is implemented in C++. The input sheet music is first parsed. We work with the MusicXML format, which was originally built for easy interchangeability (Good, 2001). The open source XML Parser of dr. ir. Frank Vanden Berghen (Vanden Berghen, 2013) is used to read the files. The parser reads all piano notes in the piece, entered by the user of the program. This information is then stored into a vector of objects called 'notes'. They contain a unique ID, the pitch, the start slice of the note, the duration (in number of slices), the used hand and the fingering, as shown in Figure 5. The pitch of the note is represented by a number, that is odd for white and even for black keys. This number resembles the midi pitch, but is still different due to the added 'fake' black keys. The first C on the keyboard is attributed the value of 15. Numbering goes on in a way that every following pitch is attributed a value that is one higher than the previous, also giving a number for the non-existing black keys between E and F and between B and C, as described in Section 3.2.

Figure 5: Visualisation of the class 'note'.

Note
+ ID + Pitch + Start + Duration + Hand + Finger

In order to store the temporal information of each note in an easily retrievable and sequential way, a second data structure is created, a vector of vectors that are called 'slices', used to represent the smallest time discretisation in a piece. This structure stores the ID's of the active notes in each slice. A zero is used to represent a rest. The combination of these two data structures allow the algorithm to make the required calculations.

During these calculations, for each move in a neighbourhood, the algorithm has to calculate the score difference for the objective function caused by this move. That score difference contains the required information on how much the objective function increased or decreased. Because of the nature of the objective function (see Section 3.2), score differences are only found in parts where changes are performed. By subtracting only the subscores for these parts instead of the entire scores, the improvement information is obtained much faster.

After the execution of the algorithm, a new MusicXML file is generated, containing the same information as the input file, but with a fingering node added to all piano notes, containing the calculated fingering. This file can be opened using music notation software like the open source MuseScore[4]. A plug-in is provided that saves the current file as an XML, runs the algorithm on it and opens the generated output.

6. Experiments

The algorithm, discussed in Section 4, brings up the need for some decisions concerning the parameters that have to be set for optimal performance. Two experiments were conducted to base the choice of these parameters on a thorough statistical analysis. Section 6.1 discusses the experiment conducted to determine the parameters 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. This is discussed in Section 6.2. It is possible to set these parameters independently, with the results of the first experiment implemented in the second. By supposing independence of these two parameter groups, the number of runs and thus computing time decreases exponentially. This approach is similar to that of Palhazi Cuervo et al. (2014) in their experiments, focusing on a subset of parameters. Both the experiments were run ten times on ten pieces of one to three pages in sheet music notation. The computer used for the experiments, was a 2.93 gigahertz Intel Core i7 processor.

6.1. Intensification phase

The first experiment to determine the optimal values of the intensification phase parameters, takes six parameters into account. For the preprocessing step `Swap`, `step_swap` and for every neighbourhood, a parameter `nbh_X` codes for the inclusion (On) or exclusion (Off) in the algorithm. The parameter `tabusize` expresses the tabu tenure as a percentage of the number of notes in the piece. A final parameter `improv_strat` indicates the use of steepest descent or first descent as the improvement strategy. An overview of the values of the parameters is given in Table 3. The impact of these parameters on two dependent variables is studied. 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` gives information on the total computing time used for one test run. A full factorial experiment was conducted on all of these variables. For every instance, $2^5 \cdot 4$ or 128 executions of the algorithm were performed. This was repeated ten times for every piece. Since we considered a benchmark set of ten pieces, a total of 12800 observations were obtained.

The runs were processed in a mixed-effects analysis of variance (ANOVA) with the statistical software JMP. The name of the piece on which an observation is done, is included in the model as a random effect and second degree interaction effects are also included. The p-values of the F-tests are given in Table 4, indicating the significance of the impact of the parameters and of

[4]This software is available from `musescore.org`.

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_swap	Off, On
tabusize	5%, 10%, 15%, 20%
improv_strat	Steepest, First

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_swap	<0.0001 *	<0.0001 *
tabusize	<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 $\alpha = 0.05$

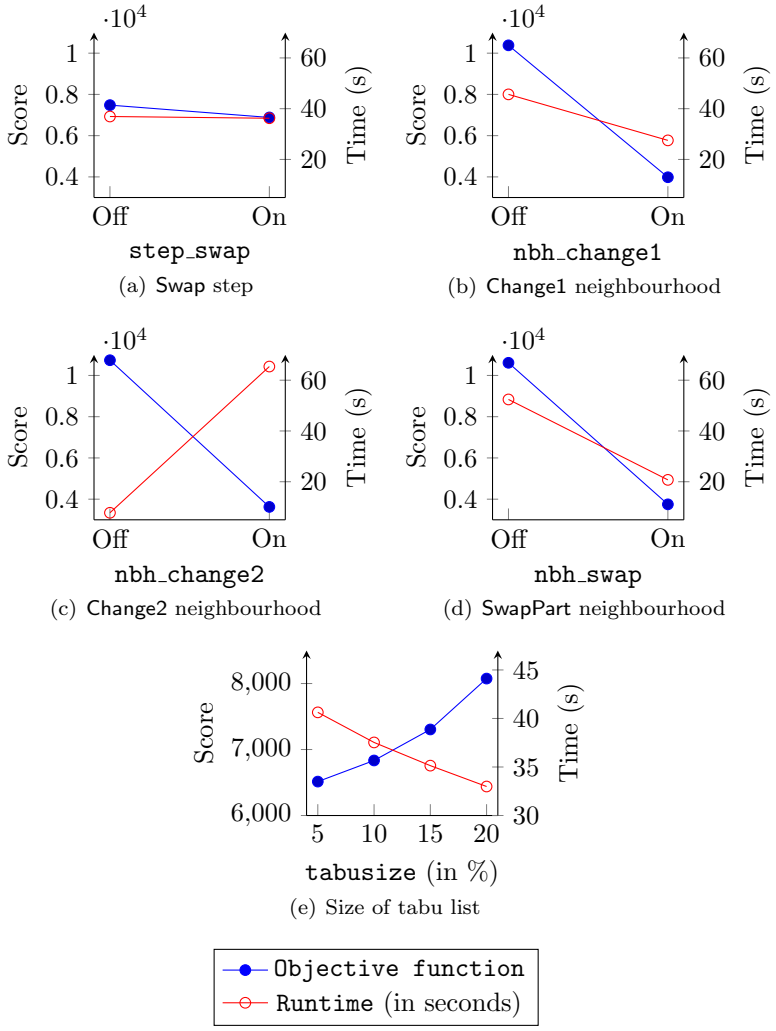
some interesting interaction effects on the respective dependent variables. It can be seen from this table that the inclusion of the preprocessing step and of each neighbourhood has a significant impact on the performance of the algorithm in terms of the `Objective function`. The mean plots in Figure 6 show that this impact is always positive, as including the step or neighbourhood reduces the score ceteris paribus. The tabu tenure also has a significant effect on the score, but the mean plot shows that a higher tabu tenure decreases the solution quality. As far as the model for `Runtime` is concerned, the p-values are also included in Table 4 and the corresponding mean plots are given in Figure 6.

A remarkable result is that the first order effect of the improvement strategy has no influence on the score, nor the runtime. When the impact of some interaction effects on the runtime is analysed (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 best strategy, but in Change2, steepest descent performs significantly better.

The graphical example in Figure 8 shows the evolution of the score over time, running the entire algorithm with steepest versus first descent. It becomes clear that first descent requires more steps, as they make smaller improvements than with steepest descent. These smaller steps however take less time, as the algorithm does not have to search through the entire neighbourhood. This negative correlation between the amount of improvement of the objective function and the amount of time used by the steps, results in the observation that both strategies consume a similar amount of time. In addition, the output of the algorithm is robust so that the improvement strategy does not influence the final objective function score significantly.

As a result from the previous analysis, we chose to include the Swap step and every neighbourhood and to deactivate the tabu list. The chosen improvement strategy is the steepest descent, as in general it appears to be slightly better and faster. An overview of these final values is given in Table 5.

Figure 6: Mean plots for the intensification phase experiment.

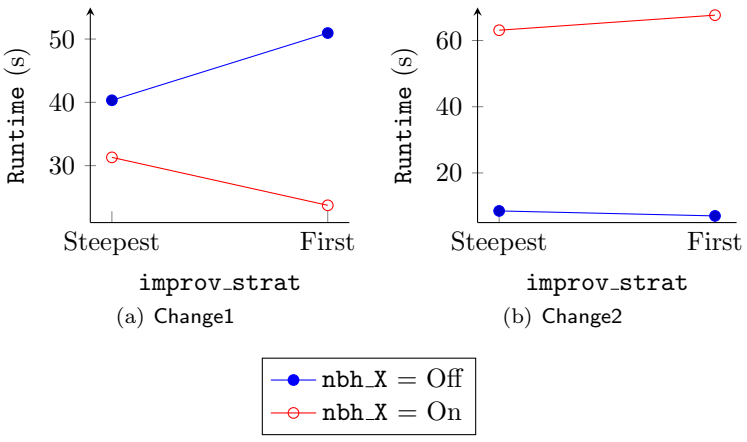


(a) Swap step (b) Change1 neighbourhood

(c) Change2 neighbourhood (d) SwapPart neighbourhood

(e) Size of tabu list

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



(a) Change1 (b) Change2

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

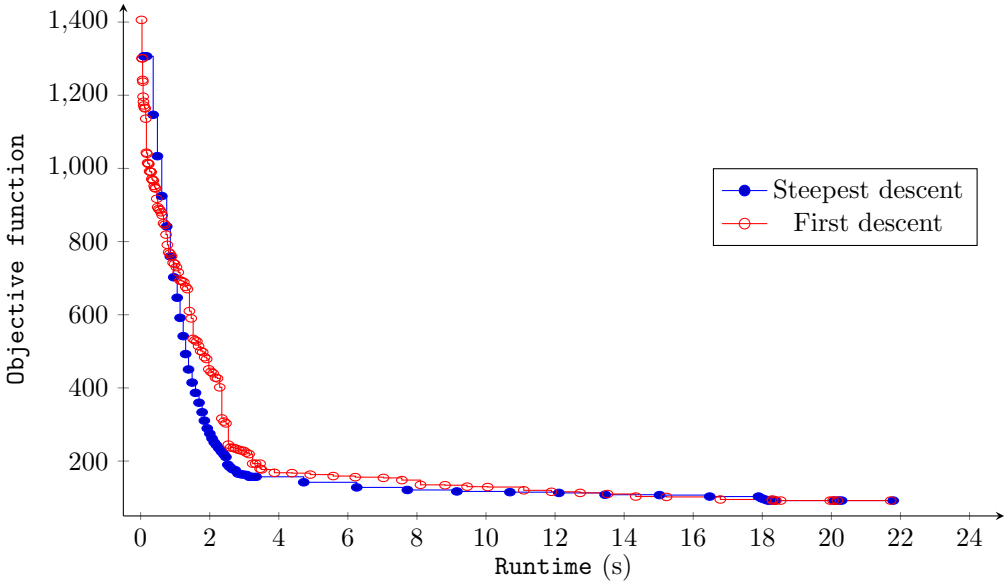


Table 5: Best values of the parameters for the intensification phase experiment.

Parameter	Value
`step_swap`	On
`nbh_change1`	On
`nbh_change2`	On
`nbh_swap`	On
`tabusize`	0%
`improv_strat`	Steepest descent

6.2. Entire VNS

With the optimal parameter settings of the intensification phase fixed in the previous experiment, we set up a second full factorial experiment with the same ten pieces to study the parameters to optimise the performance of the entire VNS. There are three parameters that have to be set. Their examined values are given in Table 6. The allowed number of iterations without improvement is given by the parameter `nrofiterations`. The amount of notes that are changed randomly are given by `sizeofpert` as a percentage of the total number of notes. The parameter `partsize` indicates the size of the parts as a percentage of the number of slices. Parts of 10% e.g. result in ten 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 ten and `sizeofpert` beyond 20 did not improve the `Objective function`. It is plausible that `Runtime` will increase at the expense of higher values for these parameters. After running all 4^3 or 64 possible parameter combinations ten times on the ten different pieces, a total of 6400 observations were obtained.

Again, a mixed-effects ANOVA was performed in JMP. We include the instance name as a random effect. Second degree interaction effects are also included again to see how the performance of the entire VNS is influenced by a combination of parameters. Table 7 shows that the three different parameters have a significant impact on the two dependent variables `Objective function` and `Runtime`. For the parameter `partsize`, the impact of the size of the parts on the dependent variables is shown in a mean plot in Figure 9. The best performance is obtained with parts of 25%, or 4 parts per piece. As far as `nrofiterations` and `sizeofpert` are concerned,

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 $\alpha = 0.05$

it is logical that the increase of these parameters will improve the performance of the algorithm by decreasing the `Objective function`, but it will also take a longer `Runtime` to execute the algorithm. Table 7 shows that the interaction effect between these two variables has a significant impact on `Objective function` and `Runtime`. Hence the combined influence of both parameters on the score and time is visualised in Figure 10. One can see that the marginal added performance of an increase in one of these two parameters is decreasing when the other parameter already has a high value. Moreover, the required calculation time would increase strongly. As a result, increasing `nrofiterations` beyond ten and `sizeofpert` beyond 20% will not increase the performance by much, only the calculation time would go up.

From the results in this analysis, the parameters that give the best performance of the algorithm are determined. The resulting values are included in Table 8. According to the experiments, the best parameter settings are ten iterations without improvement and four parts (each of 25% of the piece) of which 20% of the notes are perturbated with each iteration.

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.

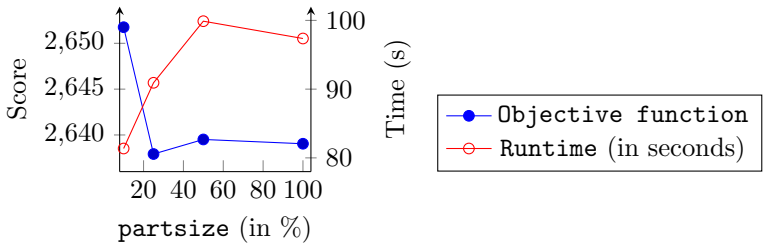
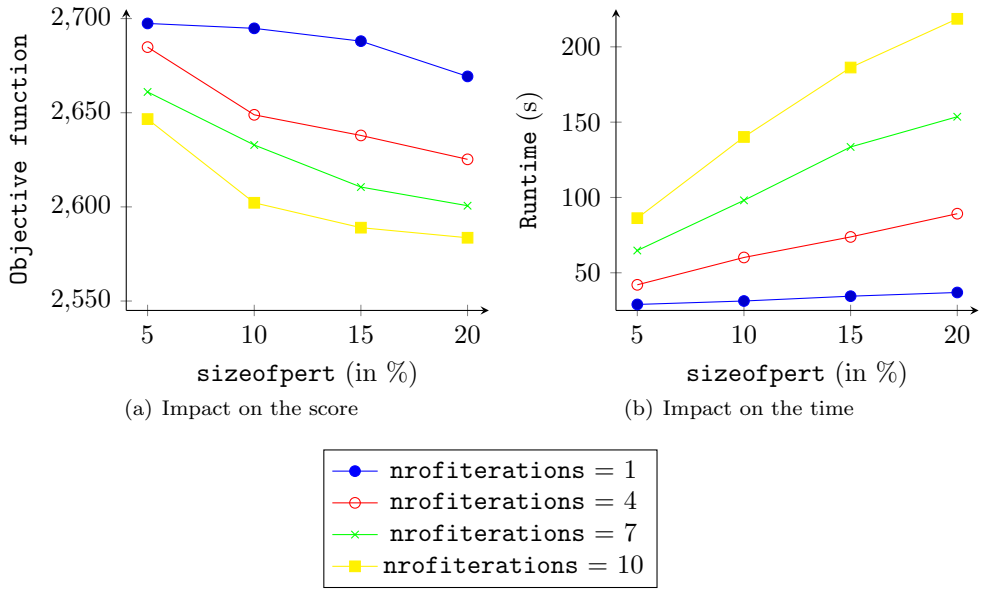


Figure 10: Interaction effect between the size of perturbation and the number of iterations for the entire VNS experiment.



(a) Impact on the score

(b) Impact on the time

7. Results and future research

7.1. Output discussion

The performance of the VNS with optimal parameters is analysed in this section. A monophonic piece that was not used in the previous experiments, namely Menuet 4 of J. S. Bach (BWV Anh. 114), was used to do a test run. The solution for the right and left hand together 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 efficiently 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 efficiency of the VNS.

The output for the first eight bars of the piece is shown in Figure 12. The bold fingerings are those that the composer or editor of a book prints on the sheets, so that the pianist automatically knows the rest of the fingering. From these notes, only one was mistaken in the sixth bar of the excerpt. In the remainder of these first eight bars, a few other small sequences require manual adaptations.

A similar analysis can be made using a polyphonic piece, namely the first variation on the Saraband from G. F. Händels Suite in D minor (HWV 437). Here, the final solution was computed in 38 seconds. The evolution of the score over time is included in Figure 13, showing the additional improvements after certain iterations. The output for the first eight bars of this piece is included in Figure 14. When we compare them again with the fingerings in the book, we see that only a few bold fingerings differ slightly. Manual adaptations can easily fine-tune this final solution, that was already fluently playable.

7.2. Future research

These minor adaptations to the final solution could be fixed 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.

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

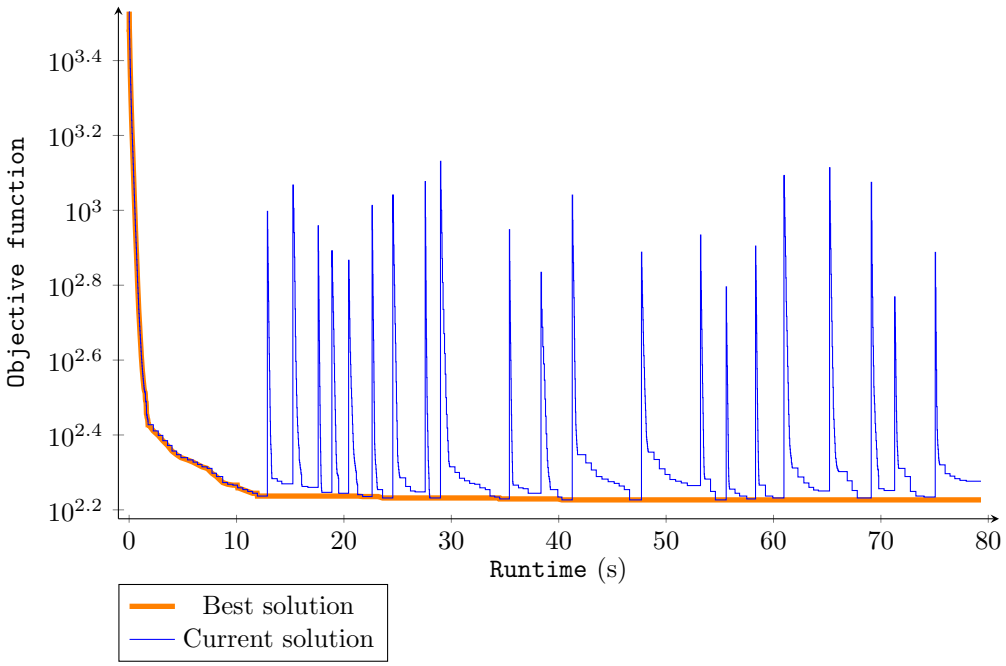


Figure 12: Output for the first eight bars of Menuet 4 (BWV Anh. 114) by by J. S. Bach.

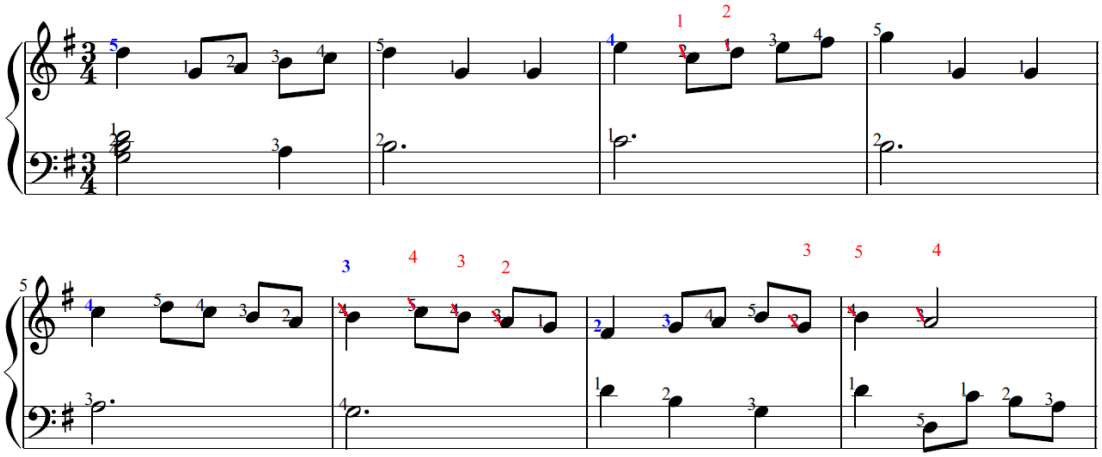


Figure 13: Evolution of the score over time in the first variation on the Saraband from Suite in D minor (HWV 437) by G. F. Händel.

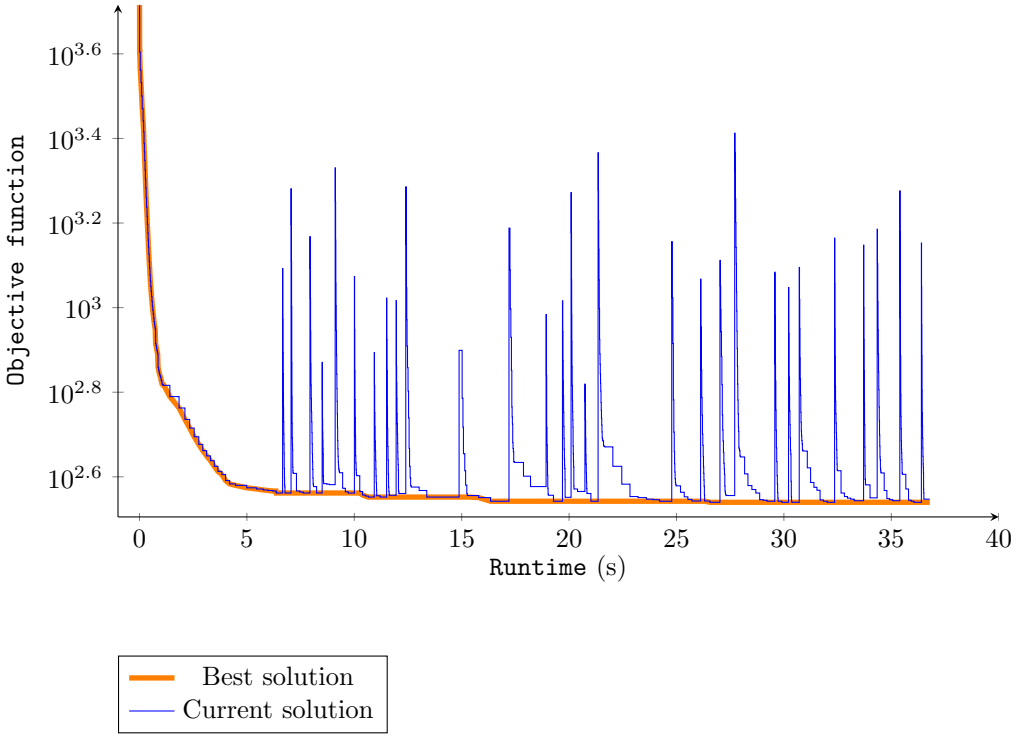
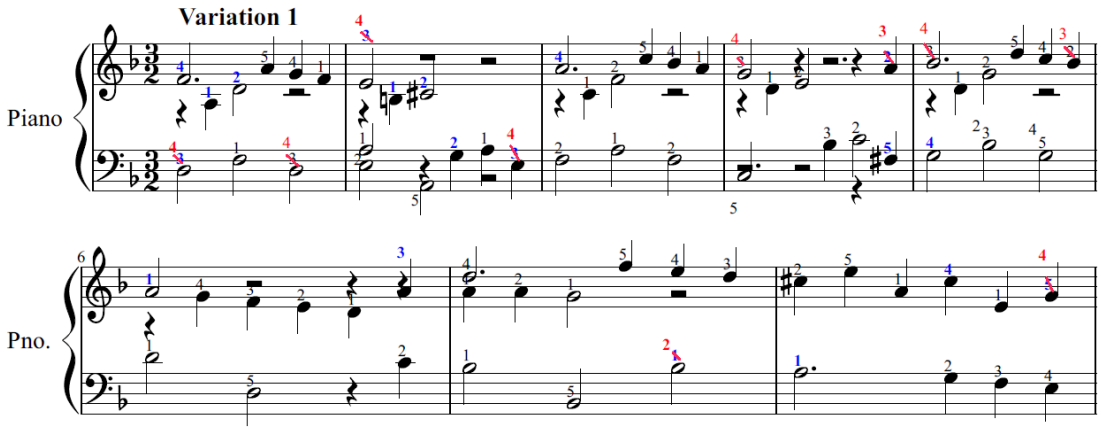


Figure 14: Output for the first eight bars of the first variation on the Saraband from Suite in D minor (HWV 437) by G. F. Händel.



7.2.1. Algorithm improvements

The algorithm might become more powerful in first place if extra neighbourhoods are added. If groups of required manual adaptations show similar characteristics, they might be converted into a neighbourhood, as long as the local modification remains small enough to guarantee a limited runtime and high efficiency.

Moreover, our VNS algorithm reads a whole piece at once, calculating one objective function for each hand in the entire piece. The calculation time could be reduced by splitting up a piece in smaller parts. A future version of the algorithm could separate a piece into parts for which an individual objective function is calculated. Such a split could be made when a rest is encountered or between two arches, indicating musical sentences. Two possible advantages that require verification are shorter calculation times due to a reduction in the number of solutions and the size of neighbourhoods on one hand and a fingering taking musical sentences better into account on the other hand.

The algorithm might also become faster by replacing the generation of a random initial solution by some greedy heuristic that takes into account the relative pitch of the notes within a slice and over the subsequent slices. These should 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 an experiment as those described in Section 6 (Palhazi Cuervo et al., 2014).

7.2.2. Objective function improvements

To obtain the best possible solution, the *set of rules* forming the objective function must be as accurate as possible to reflect that the solution is actually better. Through rule 6 e.g., the use of fingers 3 and 4 one after another is punished. This seems to lead to solutions with fragments that can be counter-intuitive for a pianist. Therefore, we advise users to set the score of this rule to a very small value, so that it would only favour this alternative in case of indifference between solutions, or even to zero and thus avoiding strange constructions as in bar 6 of Figure 12. Another approach to deal with this somewhat arbitrary score selection (Yonebayashi et al., 2007a) could be a parameter training of the different scores to make the rules as realistic as possible.

It might also be put forward that some *interpretational rules* should be implemented. One such an example is a rule 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 research 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 stressed. On the other hand, these obvious rules are only a small subset of the interpretational rules, in turn being piece, pianist and composer dependent (Newman, 1982; Bamberger, 1976).

In order to draw good conclusions on a trade-off between interpretation and easiness of a fingering, complementary research is mandatory. At first, it is advised to discern a set of interpretational rules that is as complete as possible and that might even be style-specific. In this way, the impact of interpretation on a fingering can be addressed quantitatively without drawing partial conclusions valid only for specific pieces. Secondly, this approach could be enhanced by training the values of the interpretational subscores on a sample set of pieces of a specific genre. This could avoid an arbitrary choice of the values for this essentially qualitative aspect of a piano fingering decision.

7.2.3. Extension of the research

A possible extension of our research is the quantification of the difficulty of a piece (Yonebayashi et al., 2007a) and its fingering through a parameter, taking the score of the objective function and the size of the piece into account (e.g. measured by the number of notes and the number of bars). This could be used by score analysing systems as described by Sébastien et al. (2012).

8. Conclusions

A good piano fingering is indispensable for a pianist to play a piece fluently. Based on the literature review, the problem description was defined and expanded to be able to deal with entire piano pieces of polyphonic music for the left and the right hand. By expanding the objective function and reducing the amount of hard constraints, we obtained a more realistic objective function that can also distinguish between white and black keys on the piano. User-specific finger distances and scores of the objective function rules allow personal solutions.

In this paper, we developed an efficient VNS algorithm to solve the piano fingering problem in a short amount of time. VNS is a local search algorithm that can explore different neighbourhoods. When a local optimum is reached, a perturbation is performed on the best solution and another iteration of the algorithm is executed. Two full factorial experiments allowed us to find the optimal parameters of the algorithm. We showed that all the introduced neighbourhoods have a significant added value in the algorithm.

A subsequent test run illustrated the added value of the different elements of our metaheuristic and the efficiency of the algorithm to find a very good solution for the piano fingering problem. Some suggestions to improve the algorithm and the problem description in terms of performance and speed are given subsequently. Suggestions for future research in piano fingerings, such as the expansion of the algorithm and the objective function, are also based on these results. Moreover, it might be interesting to see if interpretational rules could be added to the algorithm and if the quantification of the difficulty of a piece, based on the objective function, is possible.

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* (pp. 237–280). Columbia University Press volume 4.

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

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*, 5–23.

Glover, F., & Laguna, M. (1993). *Tabu search*. Kluwer Academic Publishers.

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*, 167–177.

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

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

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

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*, 1097–1100.

Neumann, F., & Witt, C. (2010). Combinatorial optimization and computational complexity. In *Bioinspired Computation in Combinatorial Optimization* Natural Computing Series (pp. 9–19). Springer Berlin Heidelberg.

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

Vanden Berghen, F. (2013). Kranf site: research. URL: `http://www.applied-mathematics.net/tools/xmlParser.html`.

Viana, A. B., & de Morais Júnior, A. C. (2003). Technological improvements in the siedp. In *IX Brazilian Symposium on Computer Music (August). Held in Campinas, Spain*.

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". URL: `http://hil.t.u-tokyo.ac.jp/research/introduction/PianoFingering/Yonebayashi2007IJCAI-article/index.html`.